TASK: PA12
CDRL: M001R1
21April 1995

Scenarios for Analyzing Architecture Description Languages Version 1.0

STARS-AC-M001R1/001/01

TASK: PA12
CDRL: M001R1
21 April 1995

INFORMAL TECHNICAL REPORT

For

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Scenarios for Analyzing Architecture Description Languages
Version 1.0

Accesion For	
NTIS CRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

STARS-AC-M001R1/001/01
21 April 1995

CONTRACT NO. F19628-93-C-0130

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom, AFB, MA 01731-2816

Prepared by:
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited

TASK: PA12
CDRL: M001R1
21 April 1995

Data Reference: STARS-AC-M001R1/001/01
INFORMAL TECHNICAL REPORT
Scenarios for Analyzing Architecture Description Languages
Version 1.0

TASK: PA12
CDRL: M001R1
21 April 1995

Data Reference: STARS-AC-M001R1/001/01
INFORMAL TECHNICAL REPORT
Scenarios for Analyzing Architecture Description Languages
Version 1.0

Abstract

This document describes a set of scenarios for analyzing architecture description languages (ADLs). The scenarios are designed to be used in conjunction with the ADL descriptive model framework being developed by the SEI and the CARDS program. The document includes filled-out evaluation forms for two ADLs that were analyzed using the scenarios.

TASK: PA12
CDRL: M001R1
21 April 1995

Data Reference: STARS-AC-M001R1/001/01
INFORMAL TECHNICAL REPORT
Scenarios for Analyzing Architecture Description Languages
Version 1.0

Principal Author(s):

Paul Kogut Date

Richard E. Creps 27 Apr 95

Dick Creps Date

Frank Svoboda 21 APR 95

Frank Svoboda Date

Approvals:

Teri F. Payton 4/27/95

Program Manager *Teri F. Payton* Date

(Signatures on File)

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	21 April 1995	Informal Technical

4. TITLE AND SUBTITLE

Scenarios for Analyzing Architecture Description Languages
Version 1.0

5. FUNDING NUMBERS

F19628-93-C-0130

6. AUTHOR(S)

Paul Kogut, Dick Creps, Frank Svoboda

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091-3499

8. PERFORMING ORGANIZATION REPORT NUMBER

CDRL NBR
STARS-AC-M001R1/001/01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
ESC/ENS
Hanscom AFB, MA 01731-2816

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

M001R1

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution “A”

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes a set of scenarios for analyzing architecture description languages (ADLs). The scenarios are designed to be used in conjunction with the ADL descriptive model framework being developed by the SEI and the CARDS program. The document includes filled- out evaluation forms for two ADLs that were analyzed using the scenarios.

14. SUBJECT TERMS

15. NUMBER OF PAGES

97

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Data Reference: STARS-AC-M001R1/001/01
INFORMAL TECHNICAL REPORT
Scenarios for Analyzing Architecture Description Languages
Version 1.0

Table of Contents

1.0 Introduction **1**
1.1 Architecture Description Languages 1
1.2 Characterizing ADLs 2

2.0 Overview of Activities and Results **4**
2.1 Scenario Development Approach 4
2.2 Overview of Scenarios 6
2.3 Applying the Scenarios 7

3.0 Cruise Control Scenario **9**
3.1 Scenario Objectives 9
3.2 Problem Domain Description 9
3.3 Informal Architecture Description 9
3.3.1 GM Architecture 10
3.3.2 Ford Architecture 12
3.3.3 Chrysler Architecture 13
3.3.4 Toyota Architecture 15
3.3.5 IVHS Simulator Architecture 18
3.4 Tasks and Evaluation Questions 19
3.4.1 Domain Engineer 19
3.4.2 Application Engineer 22

4.0 Command and Control Scenario **24**
4.1 Scenario Objectives 24
4.2 Problem Domain Description 24
4.3 Informal Architecture Description 24
4.3.1 Communicating Processes Architecture 26
4.3.2 Event Systems Architecture 26
4.4 Tasks and Evaluation Questions 29
4.4.1 Domain Engineer 29
4.4.2 Application Engineer 31

References **34**

Appendix A. ADL Evaluation: Capture **A-1**

Appendix B. ADL Evaluation: UniCon **B-1**

1.0 Introduction

Systematic, architecture-based reuse is a key theme of the DoD Reuse Vision and Strategy [DoD92] and a major focus of several DoD software technology programs within ARPA and the services. Domain architectures are considered one of the key technical foundations for systematic reuse. Yet architecture representation to support reuse remains a critical technical issue to overcome.

A number of DoD software technology programs are addressing these difficult architecture representation issues. For example:

- The STARS demonstration projects are each using distinct techniques for representing and applying domain architectures to support reuse, reflecting distinct interpretations of the STARS concept of megaprogramming.
- The Air Force Comprehensive Approach to Reusable Defense Software (CARDS) and Portable, Reusable, Integrated Software Modules (PRISM) programs have applied specific architecture representation approaches in the command center domain (e.g., to support the CARDS Command Center Library and prototype system composition capability).
- The SEI Software Architecture Technology Initiative (SATI) is conducting a number of projects exploring the representation and use of software architectures, from both a research-oriented and practical perspective.
- The ARPA DSSA, Prototech, and Software Composition programs are exploring architecture representation and tool support issues and developing prototype technologies.

1.1 Architecture Description Languages

An architecture model captures part of the knowledge about an architecture for a single system or a family of systems in a domain (i.e., a generic architecture or domain-specific software architecture). A model, by definition, is incomplete and imperfect. An architecture description language (ADL) is a set of notations, languages, standards and conventions for an architecture model. An ADL defines a set of notations (e.g. diagrams, formal languages, natural language text fields) for each view that the ADL includes. Architecture models provide one or more views of an architecture. Views highlight certain types of information and hide other types. Examples of well-known architectural views include:

- data flow diagrams
- flow charts/control flow diagram
- data models and entity-relation diagrams
- structure charts
- computer software configuration items diagrams (DoD-Std-2167A)
- module guides [PWC85]
- object-oriented hierarchy diagrams
- state and state transition diagrams
- process timing models
- memory allocation charts
- constraint networks

Other information that can be recorded in an ADL includes:

- text descriptions of reusable components,
- text discussions of rationales and tradeoffs,
- interface specifications (syntactic and semantic)

ADLs are often supported by tools. These include:

- tools for creation, modification, and browsing of architecture models
- tools for refining and composing systems
- tools that generate components
- tools that analyze and simulate a system before it is built

There are a variety of ADLs emerging from various industrial and academic research groups. For example, UniCon (language for Universal Connector support) is being developed at Carnegie Mellon University to explore issues of abstractions for architecture and composition of systems [Shaw94c]. Some ADLs are commercial products like UNAS/SALE (Universal Network Architectural Services/Software Architects Life-cycle Environment) which was developed by TRW and marketed by Rational [Krutchen94]. Other ARPA sponsored ADLs include LILEANNA [Tracz93], Rapide [Luckham93], MetaH [Binns93], and ArTek/DADSE [Terry94]. Please consult the World Wide Web-based *Software Architecture Technology Guide* (another product of this task) at URL http://www.stars.reston.unisysgsg.com/arch/guide.html for more information on these and other ADLs.

ADLs vary widely in terms of what architecture styles they support and what forms of analyses they permit. Like other tools, there is no one ADL that best fits all possible situations.
An ADL should chosen based on its suitability for the problem domain. For example, in some domains, complex sequences of states and actions are common, so the ADL should facilitate the representation of these applications.

A variety of groups use ADLs. Each group has a different perspective of what a good ADL should be. People involved in the acquisition of software specify ADLs for procurements and evaluate architecture models (encoded in an ADL) that are included in proposals and presented at design reviews. ADLs are important tools for software developers who design, build, or re-engineer application systems. ADLs are even more important for domain engineers who are responsible for developing domain-specific architectures.

1.2 Characterizing ADLs

In order to help users choose an appropriate ADL, a descriptive model framework for ADLs [KC95,CK95] is being developed by the SEI and the CARDS program. The goal is to answer the following questions:

- What features do ADLs have?
- What features help describe the differences between ADLs?

The features are organized in a taxonomy comprising system-oriented, language-oriented, and process-oriented categories. The framework document contains a form for recording the results of analyzing an ADL in terms of these features. In trying to use the form, the framework developers found it difficult to objectively analyze an ADL for some features without a context. For example, it can be difficult to determine the level of support an ADL has for a certain architecture style without trying to apply the ADL to that style. The need for hands-on, empirical methods to complement the descriptive framework and evaluation form became evident. One of the primary objectives of STARS task PA12 was to address this need.

A key task goal was to define a set of model problems for software architecture and develop scenarios for applying ADLs to the model problems. As part of each scenario, the ADL user is prompted to fill in parts of the evaluation form based on how well the ADL supports particular features defined in the descriptive model framework. To validate the scenarios, they were applied on two ADLs and the results were recorded using the evaluation form. Section 2 of this document summarizes the task activities and results, including the approach taken in developing the scenarios. The scenarios are described in detail in Sections 3 and 4. Appendices A and B contain the filled-out evaluation forms resulting from applying the scenarios to the two ADLs.

2.0 Overview of Activities and Results

2.1 Scenario Development Approach

There have been a number of efforts in recent years to develop methods for evaluating a variety of software engineering technologies, including:

- Ada environments [Weiderman86]
- software project management tools [FS88]
- software process description languages [Rombach91][KR91]
- process centered frameworks [Christie94]
- software architectures [KBAW94]
- maintainability of software systems [BCC95]

These methods are based on scenarios that are rooted in model problems relevant to the technologies being evaluated. Many of the methods were developed at the SEI and are based on a common evolving approach that was tailored to particular technologies. This approach is based on the following premises:

- Evaluations should be based on the results of well-defined experiments to maximize their objectivity and repeatability. These experiments are meant to be performed by an unbiased party.
- Methods should clearly define the scope of the functionality to be evaluated and should focus on core functionality that will be broadly relevant to the evaluated technologies and of greatest interest to the evaluators.
- Evaluation methods should be based on general user activities rather than detailed tool operations.
- Methods should be as independent of the technologies to be evaluated as possible to minimize potential biases.
- Methods should be extensible to accommodate additional user activities and experiments.

Our work on ADL scenarios was heavily influenced by the methods for evaluating languages (ADLs are languages), tools (ADLs are supported by tools), and process centered frameworks (ADLs have multiple user perspectives).

There were several major issues that needed to be resolved in order to develop the scenarios:

- How many scenarios should be developed?
- Which model problems should form the basis for the scenarios?
- How much detail should be given in the model problem description?
- How can bias for or against specific ADLs be minimized?
- How much detail should be given in the evaluation tasks/steps?
- Which user perspectives should the scenario cover?
- Which features in the descriptive model framework should be covered by the scenarios?

The answer to the first question was based primarily on the amount of resources available for the task. It was important to develop more than one scenario so that ADLs would be analyzed from multiple perspectives. Several scenarios would have been desirable, but resource limitations dictated that we develop only two.

In determining which model problems to use as the basis for each of the two scenarios, we observed that model problems which have been developed to explore software design methodologies can be adapted for software architecture [Shaw94a]. One of the more popular model problems, an automobile cruise control system, has been solved as an academic exercise using many different design methodologies, and the resulting architectures have been compared [Shaw94b]. We selected cruise control as one of our model problems because of this prior work and the fact that a number of solution architectures already exist. For the other model problem, we wanted to define a problem in a DoD application area that addresses architectures involving large-scale COTS and GOTS components. Because of our familiarity with the CARDS and PRISM work, we defined a problem involving simplified command center architectures.

The question of how much detail to include in the model problem descriptions centers around the tradeoff between defining a model problem that is specific enough to yield reasonably comparable and repeatable results, yet has little bias for or against any class of ADLs. Descriptions of model problems generally include an informal set of requirements for a system; they may also include an informal description of some aspects of the problem solution at some arbitrary level of detail (e.g., an informal diagram describing the general architectural structure of the system). We could create a scenario that says, in summary:

> Given a specification of the model problem requirements, design a solution using the ADL.

This scenario would be very unbiased, but would be too poorly constrained to yield very comparable results. Note also that any scenario requiring lots of creative activity (e.g., designing a solution from scratch) could be quite time consuming, or at least the amount of time needed could be quite unpredictable. Another approach would be:

> Given a specification of the model problem and an informal architecture description represent this architecture using the ADL.

This scenario would be biased significantly by the representation and structure of the informal architecture, but would probably yield relatively comparable results and could probably be performed in a reasonably small and predictable amount of time. We chose this approach and attempted to mitigate the representational bias by designing the two scenarios/model problems to impose a variety of alternative biases that test a range of representation capabilities (e.g., support for different architectural styles). The informal architecture description includes a high level specification of components and connections along with a diagram which shows the architecture structure. More detail may need to be added later based on experience in applying the scenarios.

Scenario-based evaluation methods have varying degrees of detail in their tasks/steps. The scenarios that focused on tool evaluations tend to have more detailed steps whereas the scenarios for

evaluations of languages tend to be less detailed (e.g. "represent this" vs. "draw component 1 then..."). ADLs are languages supported by tools so the detail in the tasks/steps tends to be at a higher level except in cases where there is a need to probe a certain tool capability. As in many of the SEI methods, questions are interspersed with the tasks to solicit information about the presence of absence of specific features.

Some scenario-based evaluation methods explicitly address multiple user perspectives. For example, Christie's methods for evaluating process centered frameworks has tasks and questions for both the process model developer and the end-user [Christie94]. We chose to adopt two user perspectives; domain engineer and application engineer. The domain engineer uses the ADL to create the architecture model. The application engineer uses the architecture model recorded in an ADL to develop a system. Both are roles defined in STARS reuse processes [STARS93].

The last issue is choosing features from the descriptive model framework that will be addressed by the scenarios. Features were prioritized in terms of how difficult they are to analyze objectively outside the context of a model problem and how practically relevant they are to ADLs currently available for evaluation. Examples of the ADL features we have chosen to emphasize are:

- Support for various architecture styles
- Ability to represent architectures of various categories of systems (e.g., real-time, distributed)
- Understandability of architectures described using the ADL
- Modifiability of architectures described using the ADL
- Support for explicit representation of variability within an architecture description to facilitate reuse
- Scalability to support large-scale architectures or components

Each scenario lists all the features which it covers. The definition and application of the scenarios will lead to refinements in the descriptive model framework especially for features in the area of understandability and variability.

2.2 Overview of Scenarios

Two scenarios which cover many of the important features in the framework have been developed. One scenario addresses the automobile cruise control domain and emphasizes architecture model creation for a variety of different architecture styles and heterogenous mixtures of styles. The other scenario addresses the military command center domain and emphasizes architecture refinement, application building, variability and modifiability with a COTS-based generic architecture. There is a certain amount of unavoidable overlap between the two scenarios, but a benefit is that the overlapping features are evaluated in two different model problem contexts. The cruise control and command center scenarios are described in detail in Sections 3 and 4, respectively.

The cruise control scenario was chosen because cruise control is a model problem for which many solution architectures, in several different styles, already exist [Shaw94b]. This scenario covers traditional architecture issues assuming custom built components. The scenario includes four

informal descriptions of cruise control architectures, each having a different architecture style. The four styles represented are:

- main program/subroutine
- pipe and filter
- object-oriented
- state-based/real-time

The four cruise control systems are then integrated into an object-oriented simulator architecture that is a heterogeneous mix of styles. The heavy emphasis on representing a variety of architecture styles may be premature based on the current state of the art in ADLs but we feel this approach will help reveal many strengths and weaknesses in ADLs.

The command and control scenario is meant to address issues of "megaprogramming" (central to the DoD Reuse Vision and Strategy) wherein architectures must support the use (and reuse) of large-scale COTS and GOTS components. One issue is the ability to handle variability, refinement and application building (i.e., system composition) in a generic architecture for a domain or product line. The second issue is connecting large pre-existing components that were not originally designed to fit together. We have chosen two architecture styles that are popular approaches to the connection problem; communicating processes and event systems. The first informal architecture description specifies remote procedure calls (RPCs) as defined in OSF's Distributed Computing Environment standard [OSF93] [Shirley93]. The second informal architecture description specifies the Common Object Request Broker Architecture (CORBA) standard [OMG91]. The important differences are that the communicating processes architecture has synchronous point to point connections whereas the event system architecture has asynchronous broadcast connections. This scenario may also be somewhat ambitious for currently available ADLs but it provides a goal for ADL researchers.

2.3 Applying the Scenarios

To validate the scenarios and produce concrete examples of ADL evaluations using the scenarios, a key aspect of the task was to apply the scenarios in a hands-on manner to a small number of ADLs (and their supporting tools) and record the results. Several toolsets supporting ADLs were solicited from their suppliers for installation and use at a STARS facility. Since many ADL support tools are currently research prototypes, the suppliers were typically the tool developers in research organizations, but some commercial vendors were also contacted. Some of the tools could not be obtained in time to perform an evaluation, while others included components requiring license fees that were not budgeted in the task. In the end, we selected two ADLs/toolsets for analysis:

- Capture, from CTA, Inc.
- UniCon, from the Carnegie Mellon University's Composable Systems Group.

Capture is designed to support legacy system management and domain analysis. It enables the architectures/designs of families of systems and subsystems to be described using a variety of

different notations. The descriptions within a family can be readily compared, contrasted, and evaluated for reuse or to support design decisions in system development and maintenance efforts. Capture will soon be offered as a commercial product by CTA.

UniCon (language for *Uni*versal *Con*nector support) is a prototype language and toolset that has been developed at CMU to explore research issues involving the formal representation of a core set of software architecture constructs and architecture styles. UniCon treats both the software components of an architecture and their interconnections as first-class architecture constructs. The toolset enables both textual and graphical architecture descriptions, both of which were used during our analysis.

The evaluation forms describing the results of applying the scenarios to these ADLs are included in Appendices A and B. The forms themselves provide significant information about the features that are defined as part of the SEI/CARDS descriptive model framework, but some readers may need additional information about the framework in order to adequately interpret the results. The best source of this information is the draft technical report describing the framework [KC95]. It has not yet been formally published, but is available in postscript form via the World Wide Web *Software Architecture Technology Guide* (http://www.stars.reston.unisysgsg.com/arch/guide.html) and may be obtained in hard copy form from Paul Clements of the SEI to a limited degree.

DISCLAIMER: It is important to note that the information recorded in the forms has not been reviewed with the ADL/tool suppliers. The forms may reflect misinterpretations or incorrect assumptions on our part that we have not had an opportunity to identify or correct. The primary function of the forms in this document is to provide examples of how the scenarios can be used in conjunction with the SEI/CARDS framework to analyze ADLs. Although we believe the forms contain useful information about the specific ADLs we have analyzed, the data should not be treated as definitive and should be corroborated before being used as the basis for ADL-related decisions.

3.0 Cruise Control Scenario

3.1 Scenario Objectives

The automobile cruise control scenario emphasizes architecture model creation and the interpretation of the resulting model. The cruise control model problem represents a more traditional single system development approach, in contrast to the domain-specific product-line oriented development approach that is found in the command and control scenario. The scenario helps determine the applicability of the ADL for a variety of different architecture styles (including a heterogenous mixtures of styles) and system categories. The scenario probes the fundamental language characteristics of the ADL including how the language captures and communicates information and what the ADL assumes about intended users. Process guidance and tool support for an ADL are also covered. The scenario has a special focus on validation of the architecture model that was created. In the later stages of the scenario there is a special step that analyzes an ADL's capability to compose large pieces of the architecture into another architecture. The specific ADL feature evaluation objectives of the cruise control scenario are found in section 3.4. Questions about the usability of the ADL are added to augment the features from the descriptive model framework.

3.2 Problem Domain Description

There are several different car manufacturers who (hypothetically) each designed their digital cruise control system based on the version of the cruise control problem described by Booch [Booch86].

- The GM cruise control architecture has a main program/subroutine style.
- The Ford cruise control architecture has a pipe & filter style.
- The Chrysler cruise control architecture has an object-oriented style.
- The Toyota cruise control architecture has a state-based/real-time style.

The Department of Transportation wants to build an object-oriented intelligent vehicle highway system (IVHS) simulator that will allow experimentation with different planning and control algorithms. IVHS keeps traffic moving and avoids accidents on a superhighway. The IVHS must incorporate each of the digital cruise control architectures so that it can simulate control signals to the cruise control systems in a car and predict the effects of these signals while planning.

3.3 Informal Architecture Description

The purpose of this description is to specify basic components and data flow so that the architectures described in various ADLs will be comparable and not need to be designed from scratch for each ADL. The various designs below are compared in [Shaw94b]. The basic characteristics of architecture styles are described in [GS93].

The informal architecture descriptions are meant to be ADL-independent so they must maintain a fine balance between being informative and holding a bias for or against specific ADL characteristics. Therefore the description includes:

- An architecture style description that implies semantics for the component and connection types
- A list of components, their component types, and a brief description
- A diagram that shows how components are connected. If no connection is shown, it should be assumed that no connection exists. The diagram may include a key to assist in interpreting the diagram and to identify connections that do not conform to the architecture style (if any).

3.3.1 GM Architecture

The GM architecture (main program/subroutine style) uses Booch's functional design [Booch86]. This style is based on the common subroutine calling that is found in most traditional programming languages. The main program calls the subroutines in some specified sequence [GS93]. Components are:

- Cruise_Control_System (main): continuously calls each subroutine below in the order shown.
- Get_Desired_Speed (subroutine): from driver
- Get_Current_Speed (subroutine): from wheel sensor
- Get_Brake_State (subroutine): on/off
- Calculate_Throttle_Setting (subroutine): calculates value for engine throttle setting.
- Put_Throttle_Value (subroutine): adjusts the throttle setting.

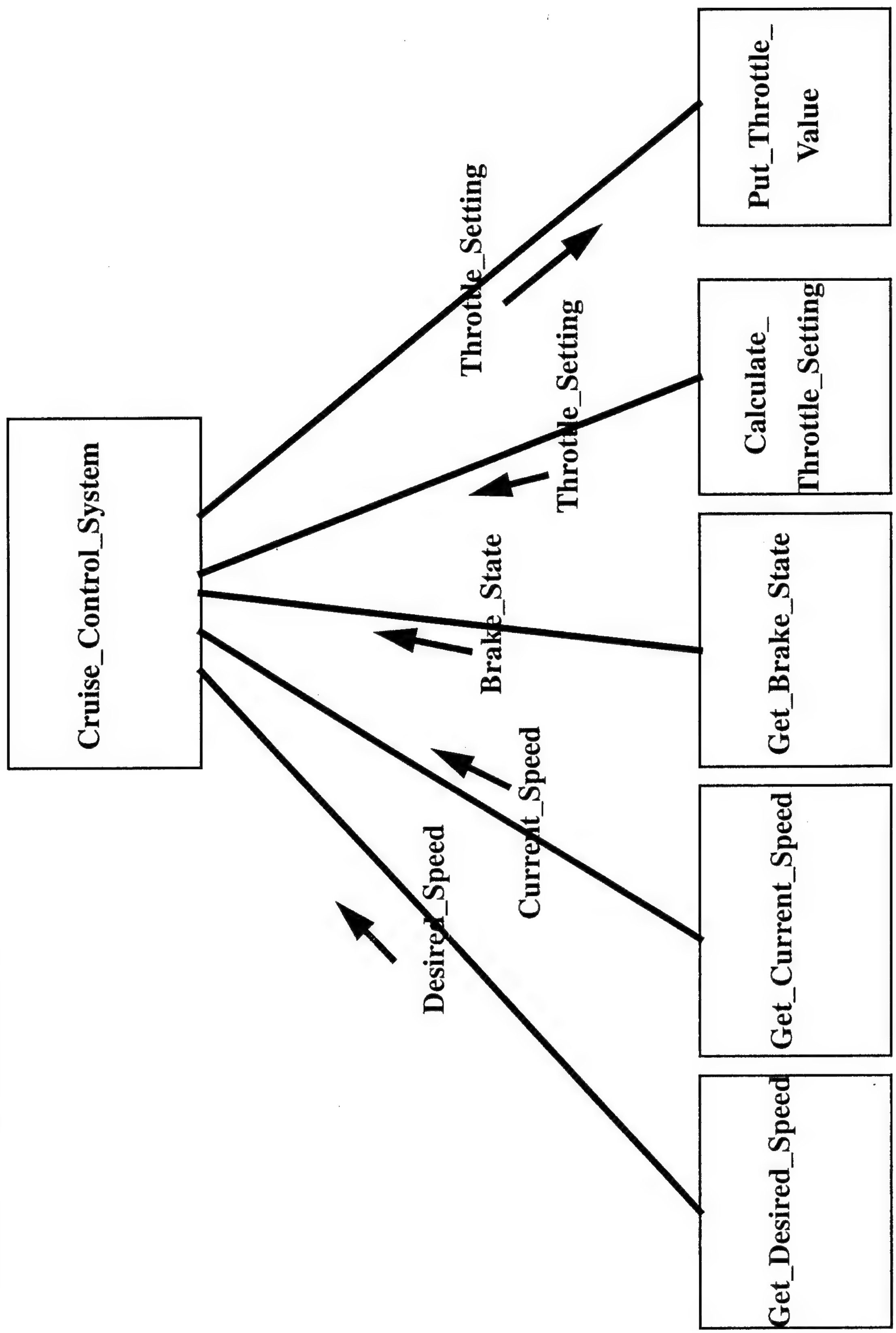


Figure 1: GM Architecture

3.3.2 Ford Architecture

The Ford architecture (pipe & filter style) is Booch's data flow diagram [Booch86] transformed into a pipe & filter architecture. UNIX pipe & filter model abstractions should be assumed. In filters output begins before the input is consumed. Filters do not share state with other filters [GS93]. Components are:

- Calculate_Current_Speed (filter): continuously outputs current speed
- Calculate_Desired_Speed (filter): continuously outputs desired speed
- Set_Brake_State (filter): continuously outputs brake state
- Calculate_Throttle_Setting (filter): continuously outputs setting to throttle

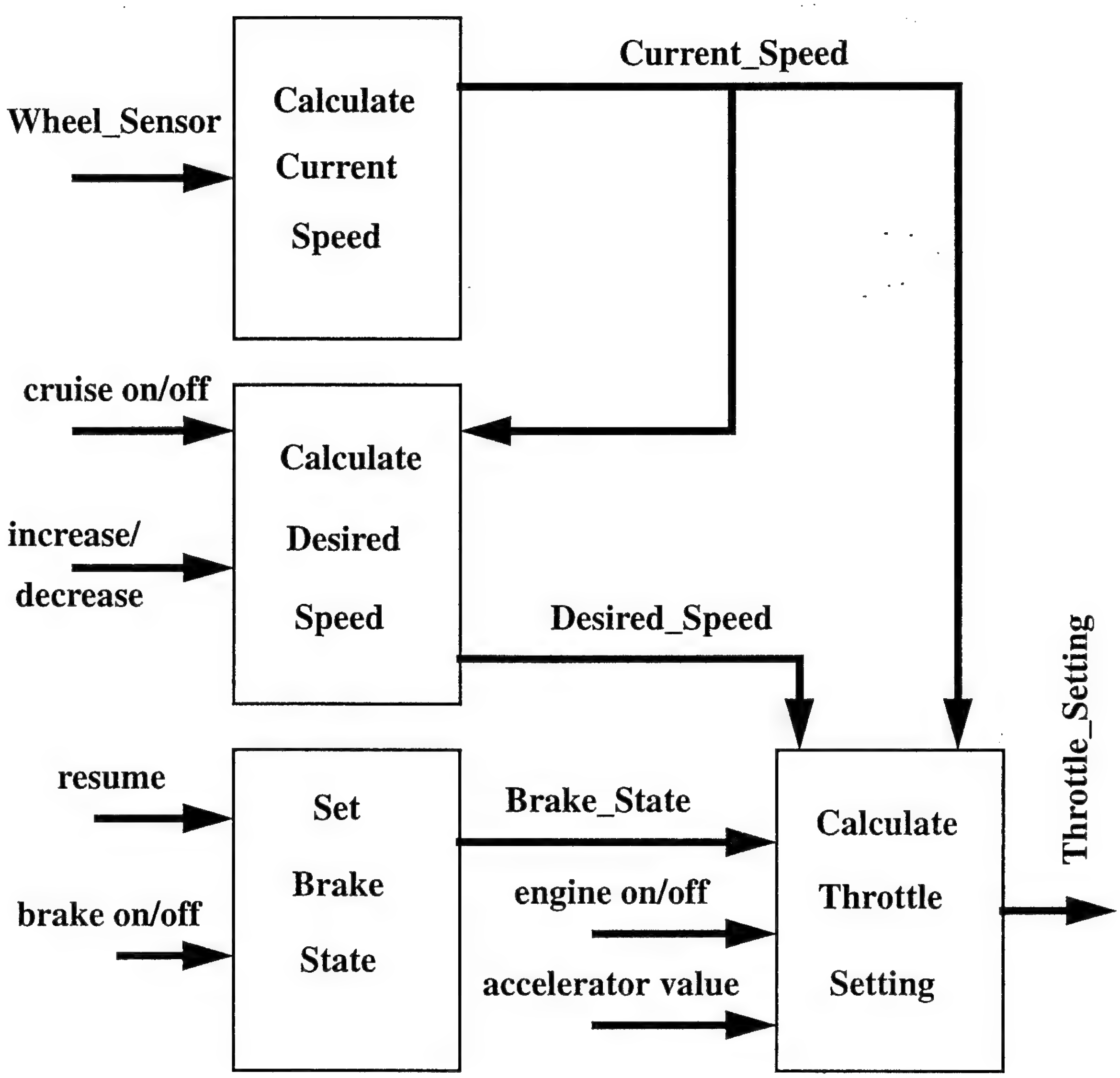


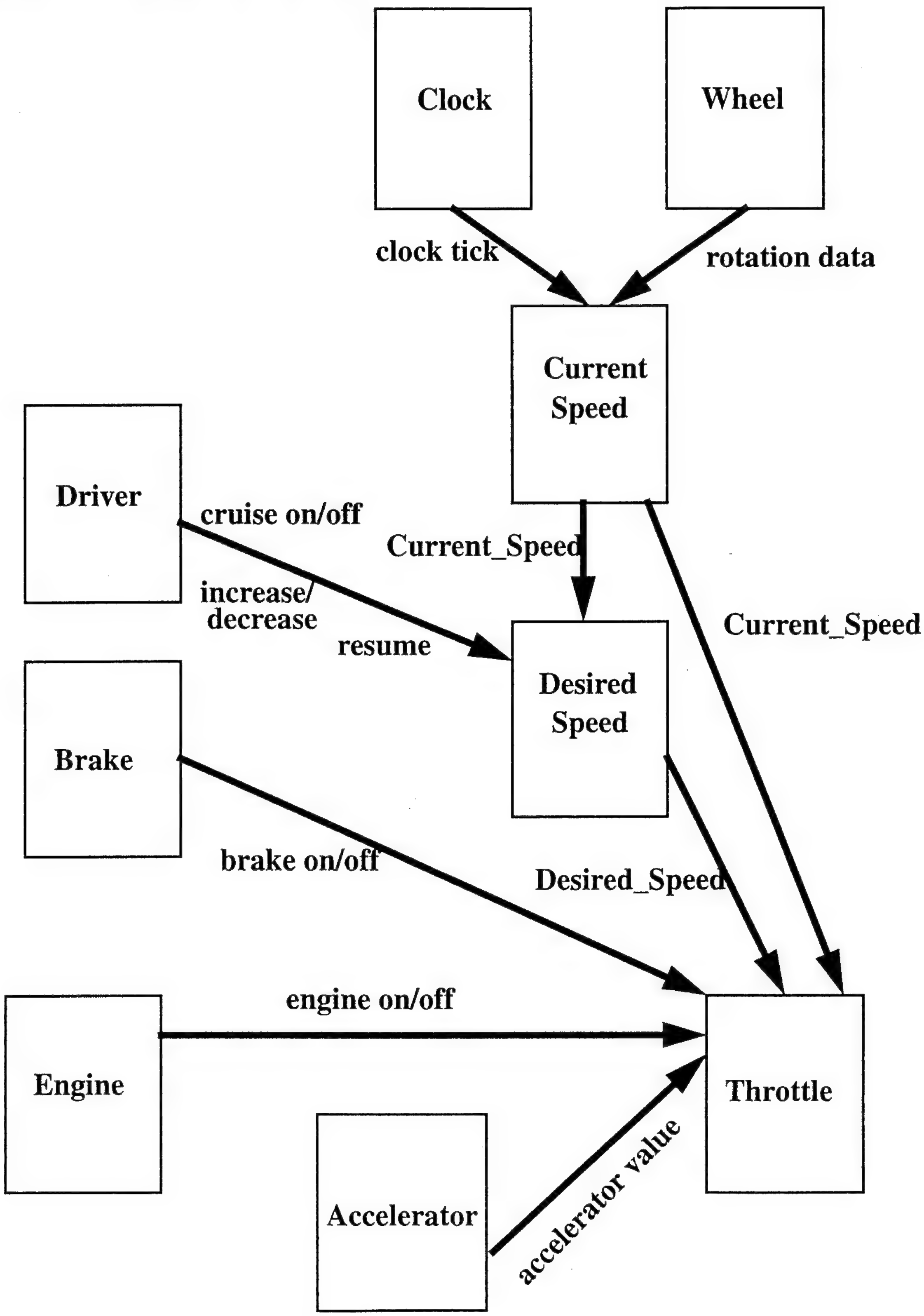
Figure 2: Ford Architecture

3.3.3 Chrysler Architecture

The Chrysler architecture (OO style) uses Booch's OO design [Booch86]. Each object in a class is an abstract data type (ADT) that has a well defined interface and preserves an internal state [GS93]. The object is an abstraction of a real world entity or some internal computing entity. The object is accessed only through its operations. Classes inherit data definitions and operations from parent classes.

- Driver (class) (parent class Agent): operations provide driver commands
- Brake (class) (parent class Device): operations provide brake state
- Clock (class) (parent class Device): operations provide timing signals
- Wheel (class) (parent class Device): operations provide rotation data
- Engine (class) (parent class Device): operations provide engine state
- Accelerator (class) (parent class Device): operations provide accelerator value
- Throttle (class) (parent class Device): operations control throttle
- Current_Speed (class) (parent class Speed): stores and updates speed of vehicle
- Desired_Speed (class) (parent class Speed): stores and updates desired speed

Figure 3: Chrysler Architecture



3.3.4 Toyota Architecture

The Toyota architecture (state-based/real-time style) uses Smith & Gerhart's activities and states [SG88]. The functional decomposition, which partitions functionality into activities, is influenced by the emphasis on states and state transitions. This scenario only includes the control speed part of the design and not the extra monitoring functions in [SG88]. Components are:

- Select_Speed (activity): stores current speed
- Clear_Speed (activity): clears speed setting
- Desired_Speed (data store): desired speed value
- Maintain_Speed (activity): maintains speed
- Increase_Speed (activity): increases speed
- Check_Speed (activity): check if a previously set speed has been reached
- Speed_Control (control activity): senses state and issues commands to other activities
- Test_Pedal_Deflection (activity): test deflection of acceleration pedal

States for Speed_Control are:

- Initial (state): state immediately after driver turns cruise control on (no previous speed setting)
- Accelerate_New_Speed (state): driver is pushing accelerator button (not gas pedal)
- Resume_Cruise (state): resume last previous cruising speed
- Cruise_Activated (state): cruising speed is being maintained
- Cruise_Inactivated (state): cruise disengaged (previous cruising speed not cleared)
- Error (state): cruise control not responding

Figure 4: Toyota Architecture

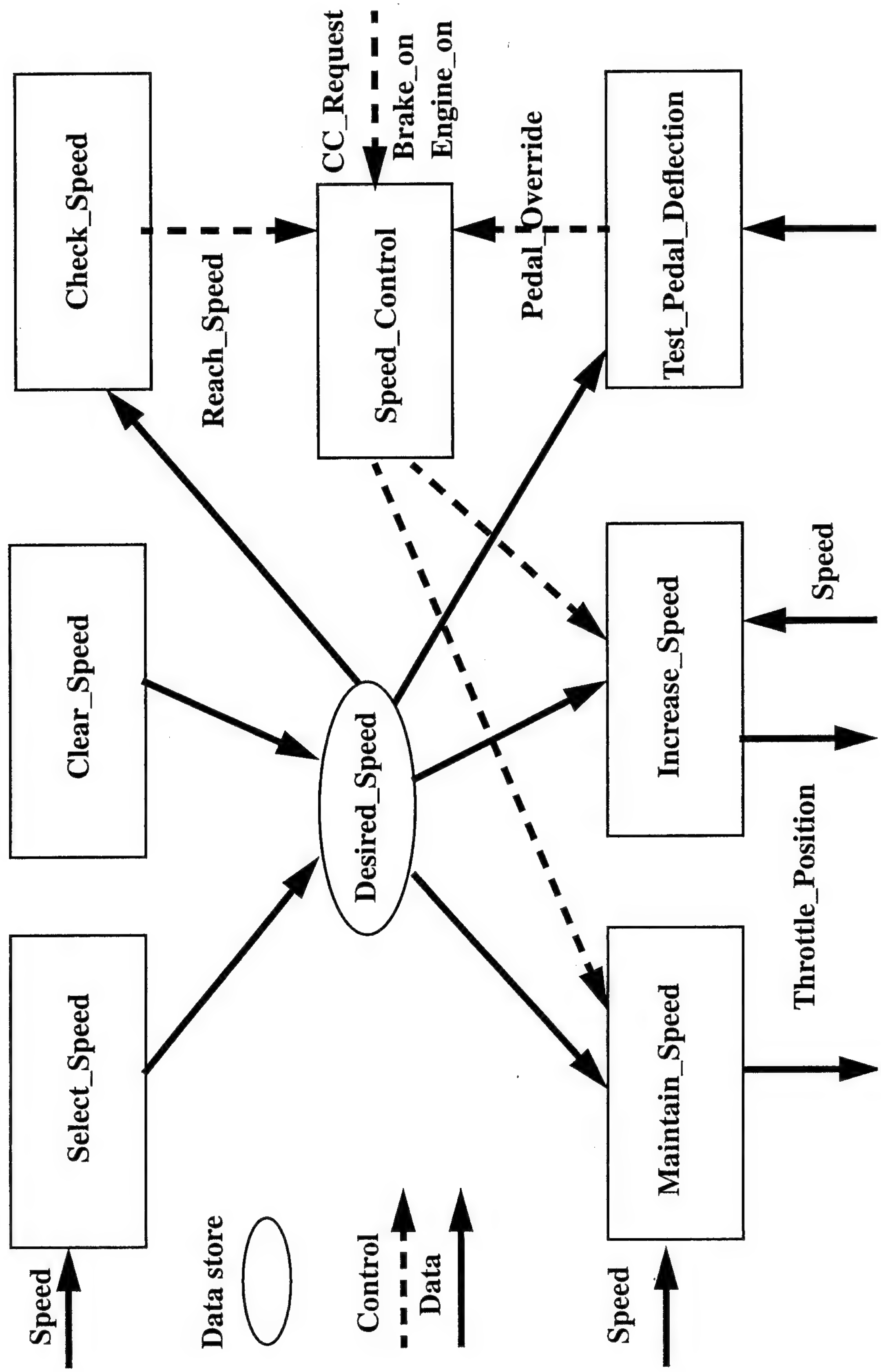
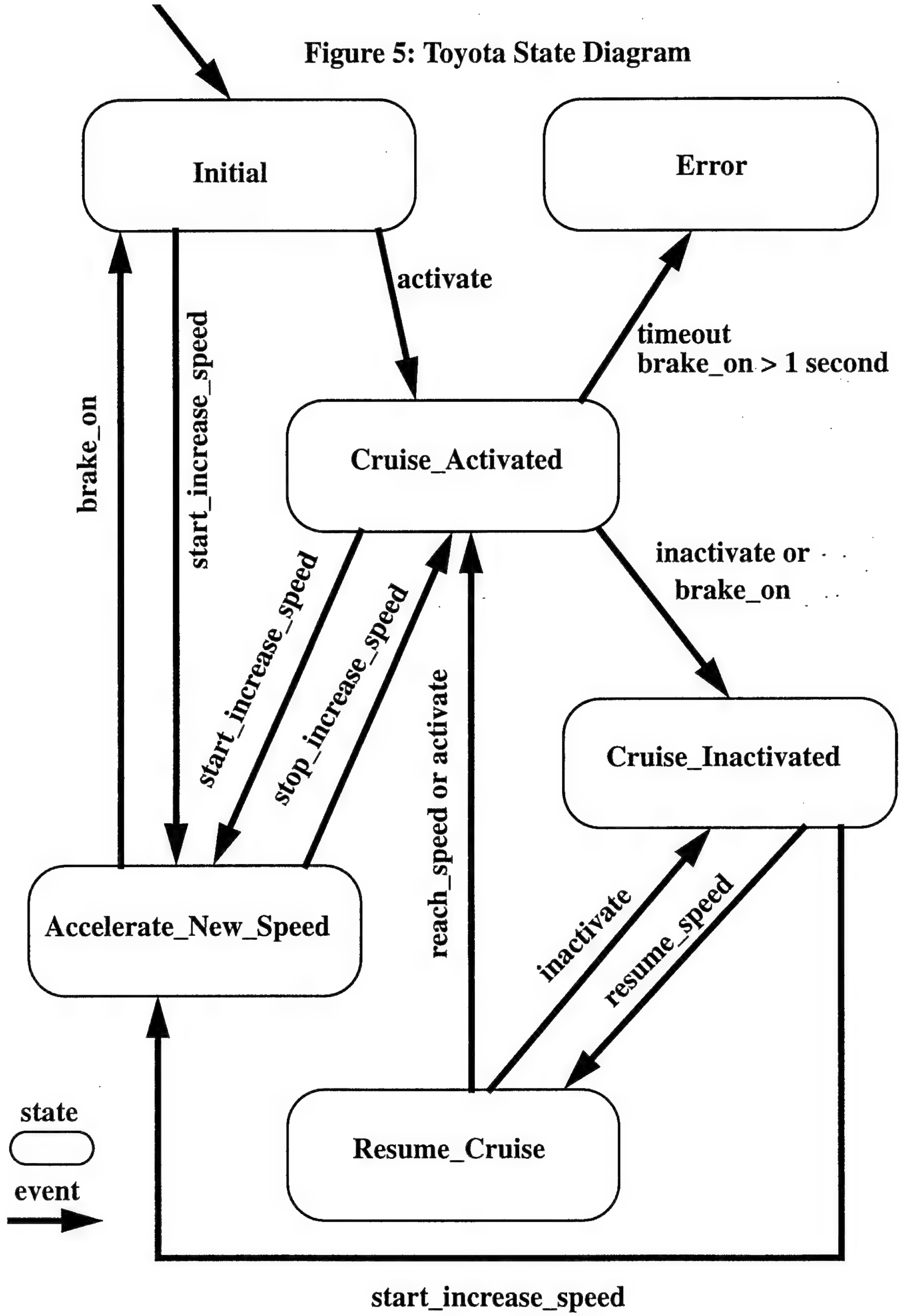


Figure 5: Toyota State Diagram



3.3.5 IVHS Simulator Architecture

The IVHS simulator design was invented for this scenario. The first four components are the cruise control systems described above. The top-level architecture style is object-oriented but the overall style is heterogeneous because the system consists of components with different styles.

- GM cruise control (class) (parent class Cruise_Control)
- Ford cruise control (class) (parent class Cruise_Control)
- Chrysler cruise control (class) (parent class Cruise_Control)
- Toyota cruise control (class) (parent class Cruise_Control)
- Traffic controller (class) - Avoids collisions. Maintains distance between vehicles by controlling their speed. Allows addition and removal of vehicles from the simulation. Reports the current state of traffic and records traffic patterns.
- Traffic planner (class) - Plans the steps taken by the controller and checks their safety.

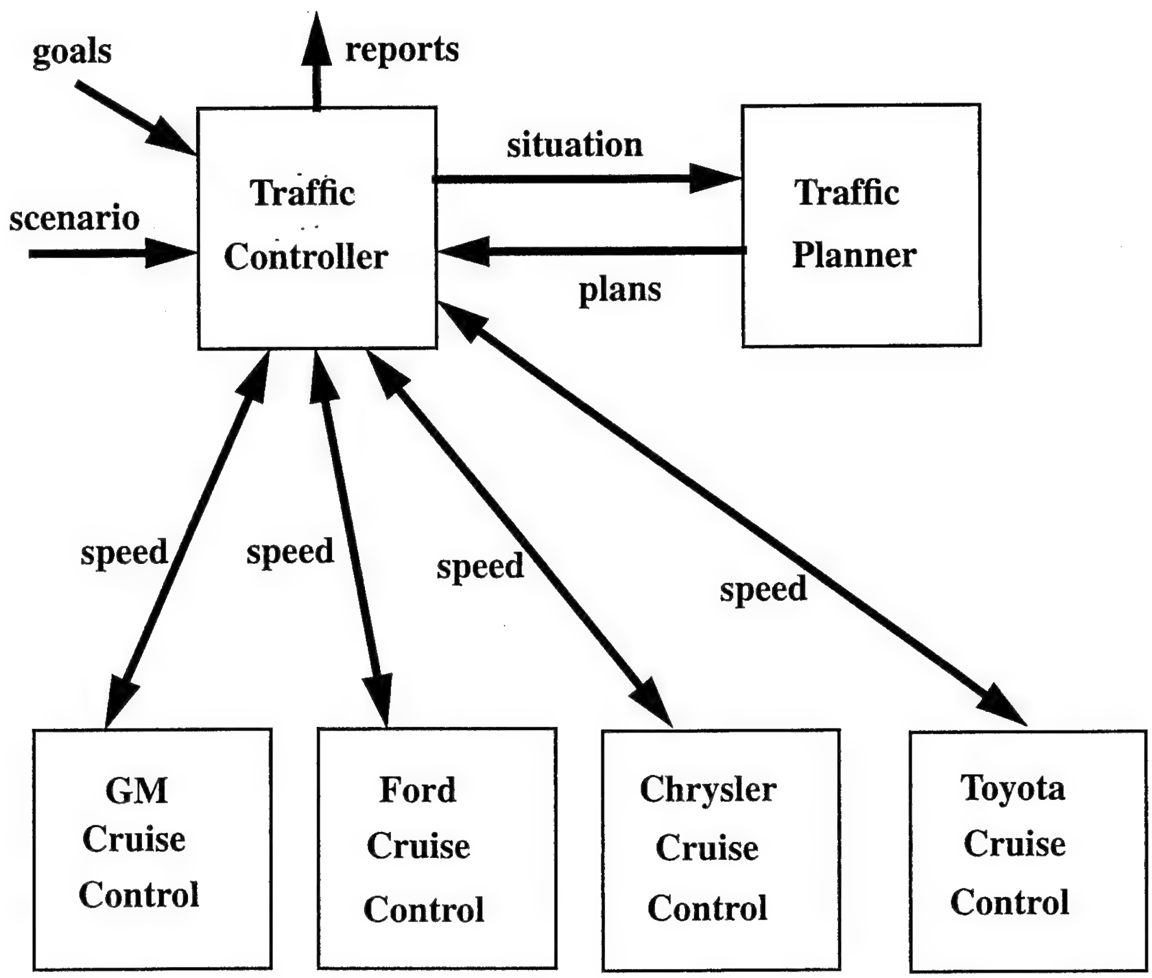


Figure 6: IVHS Simulator Architecture

3.4 Tasks and Evaluation Questions

The following tasks are divided between two main types of users; domain engineers and application engineers. Most questions refer to the numbered features in the ADL descriptive model framework (all numbers in parentheses refer to features in [KC95]).

3.4.1 Domain Engineer

The domain engineer creates the architecture model.

3.4.1.1 Create Architecture Models for Cruise Control Systems

Use the ADL to describe the architecture for as many of the four cruise control architectures/ styles as possible (i.e., either the ADL was explicitly designed to support this style or the ADL can support this style but provides few or no built-in features to do so). Use all process support materials available including process definitions, user manuals, and training course materials. Use multiple views, if they are available and applicable.

Complete the following parts of the ADL descriptive model framework evaluation form. Each column in the following tables indicates a level in the feature hierarchy.

	Process Support (3.3.7)		**process definition**
			users manual
			training course

Process-oriented (3.3)	**architecture creation support (3.3.1)**		**textual editor**
			graphical editor

System-oriented (3.1)	**applicability (3.1.1)**	**architecture style (3.1.1.1)**	**main program and subroutines**
			pipes and filters
			object-oriented
			state-based/real-time

	views (3.2.4)	**multiple views (3.2.4.1)**	
		syntactic views (3.2.4.2)	
		semantic views (3.2.4.3)	
		inter-view cross reference (3.2.4.4)	

Language-oriented (3.2)	**language definition quality (3.2.1)**	**formality (3.2.1.1)**	
		completeness (3.2.1.2)	
		consistency (3.2.1.3)	
	Expressive power (3.2.2)	**primitives (3.2.2.1)**	
		extensibility (3.2.2.2)	
		abstractions (3.2.2.3 - see table after 3.2.8)	

Evaluate the following usability features of the ADL (not in the current framework) on a scale from 1 to 5 (1 = Disagree .. 5 = Agree) [Christie94] [FS88]:

- The ADL was easy to learn.
- The ADL tools streamlines the process of creating an architecture description.
- The ADL is clear and consistent in its use of terminology.
- The ADL tools have a consistent "look and feel" in their user interface.

- The ADL tools display available options based on the current state.
- The ADL tools provide on-line help.
- The ADL tools provide useful error messages.

3.4.1.2 Architecture Validation

Do completeness and consistency checks on described architectures.

Complete the following parts of the ADL descriptive model framework evaluation form:

	architecture validation support (3.3.2)		**syntax checker**
			semantics checker
			completeness checker
			internal consistency checker

3.4.1.3 Create Architecture Models for IVHS Simulator

Use the ADL to describe the IVHS simulator architecture.

Complete the following parts of the ADL descriptive model framework evaluation form:

System-oriented (3.1)	**applicability (3.1.1)**	**architecture style (3.1.1.1)**	**heterogeneous styles**

	Modifiability (3.2.7)	**scalability (3.2.7.2)**	**abstraction levels**
			cross referencing
			composition

3.4.2 Application Engineer

The application engineer is the end user of the architecture model. The steps below should be performed by someone who didn't do the steps in 3.4.1.

3.4.2.1 Interpret Architectures Model

Answer questions about the architecture from information in the ADL. This is an experiment to determine how the ADL communicates information.

- What are the inputs of Get_Desired_Speed in the GM architecture?
- What are the outputs of Set_Brake_State in the Ford architecture?
- What is the parent class of Wheel in the Chrysler architecture?
- What causes a transition to the Resume_Cruise in the Toyota architecture?

Complete the following parts of the ADL descriptive model evaluation form:

	architecture refinement support (3.3.3)		**browser**
			search tool

3.4.2.2 Evaluation of ADL

Evaluate the following usability features of the ADL (not in the current framework) on a scale from 1 to 5 (1 = Disagree .. 5 = Agree) [Christie94] [FS88]:

- The ADL screen layouts were easy to understand.
- The ADL was easy to learn.
- The icons for components were clearly distinguishable and easy to read.
- The icons for connections were clearly distinguishable and easy to read.
- The ADL makes effective use of color.
- The ADL is clear and consistent in its use of terminology.
- The ADL tools have a consistent "look and feel" in their user interface.
- The ADL tools display available options based on the current state.
- The ADL tools provide on-line help.

Complete the following parts of the ADL descriptive model framework evaluation form:

	readability (3.2.5)	**embedded comments (3.2.5.1)**	
		presentation control (3.2.5.2)	

	Characteristics of intended users (3.2.6)	**Target users (3.2.6.1)**	
		Expertise (3.2.6.2)	

System-oriented (3.1)	**applicability (3.1.1)**	**system category (3.1.1.2)**	**hard real-time**
			soft real-time
			embedded

4.0 Command and Control Scenario

4.1 Scenario Objectives

The command and control (C2) scenario is meant to address key issues of megaprogramming involving the use and reuse of large-scale components. The C2 scenario has more focus on application engineering aspects than the cruise control scenario but model creation is still a major task. One major issue for the scenario is the ability to handle variability, refinement, and application building (i.e., system composition) in a generic architecture for a domain or product line. The C2 architecture includes a generic component structure as well as instances of components that can be plugged into the structure. The scenario also addresses the issue of connecting large pre-existing components that were not originally designed to fit together, such as commercial-off-the-shelf (COTS) products and large components from existing government developed systems (i.e., government-off-the-shelf (GOTS) components). This constraint leads to architecture styles which can accommodate these large components. The scenario also has a special focus on modifying an architecture model from one style to another. The specific ADL feature evaluation objectives of the command and control scenario are found in section 4.4. Questions about the usability of the ADL are added to augment the features from the descriptive model framework.

4.2 Problem Domain Description

The DoD wants to develop military command centers based on large COTS and GOTS components. The DoD domain engineering team needs to use an ADL to develop a simple generic architecture from which the DoD application engineering team can quickly build command centers. The domain engineering team first creates an architecture with a communicating processes style and later revises it to have an event system style.

4.3 Informal Architecture Description

The simple generic architecture for command and control systems is based on a subset of the PRISM generic command center architecture [PRISM93]. Some of the component descriptions are based on the CARDS Command Center Library [CARDS94].

The informal architecture descriptions are meant to be ADL-independent so they must maintain a fine balance between being informative and holding a bias for or against specific ADL characteristics. Therefore the description includes:

- An architecture style description that implies semantics for the component and connection types
- A list of components, their component types, and a brief description
- A diagram that shows how components are connected. If no connection is shown, it should be assumed that no connection exists. The diagram may include a key to assist in interpreting the diagram and to identify connections that do not conform to the architecture style (if any).

The C2 architecture contains the following components:

- Geographical Information System (GIS) GOTS or COTS: A Geographic Information System (GIS) is used for presenting and manipulating information in a geographical context. In basic terms, a GIS links tabular data as found in a traditional database with the visual world of maps and charts. GISs come in basically two flavors: vector and raster. A GIS should provide the ability to add well-defined standard symbols to 2- and 3-dimensional maps, and provide tools to perform analysis on both the maps and added symbols. Typical analysis requires polygon overlay, network routing, zoom in/out. Communicating with a database is necessary to add dynamic and static data to the view of the map.

- DBMS COTS: The Database Management System (DBMS) stores and manages tables of data that the Command Center will query and modify during daily activities. The DBMS should support a relational data model and a client/server architecture.

- DataBase Broker COTS: The database broker serves as an interface between the DBMS and other applications requesting the DBMS services. Its purpose is to provide better modularization and reusability for the DBMS. The database broker receives requests for database services in "generic" or standard SQL. The request is translated into commands and functions recognized by a particular DBMS (such as Sybase or Oracle). The database broker allows other DBMSs to be "plugged" in or out without affecting the client applications that rely upon the database services provided by the DBMS.

- Message Processor GOTS or COTS: The message processor (MP) component validates the accuracy of incoming fixed-format messages against a set of predefined message templates. Those incoming messages that pass validation are then translated into SQL formatted entries that are passed on to the database broker and then the DBMS for entry into the database tables. Those messages that are determined as invalid are stored in a file, and the operator is notified of invalid message receipt. The MP provides automatic parsing, routing, and storing of selected incoming messages, both free-text and fixed-format. The MP also supports the generation of outgoing messages.

- Desktop Publisher COTS (optional component): Desktop publishing components are similar to word processors but with advanced features. Some of these advanced features include: the capability to create/edit graphic figures and images within the system, and functionality for specifying the layout of the document.

- Event Manager (component in event systems architecture model - see figure 2): This component provides inter-component communication. The event manager is sometimes called middleware or an object request broker. It addresses the requirement to provide communications between software components in a way that is compatible with POSIX and GOSIP standards. This component ensures the delivery of messages and data, provides message transfer structure, and may include some performance monitoring

- User Interface (custom component): This component is a graphical user interface for the generic command center.

The following is a list of component instances (actual components that can be plugged in to perform the functions defined by the abstract components in the architecture descriptions);

GIS:
- OILSTOCK (GOTS)
- Delorme (COTS)

DBMS:
- Oracle (COTS)
- Sybase (COTS)
- Informix Online (COTS)

Database Broker:
- Omni SQL Gateway (COTS - Sybase)

Message Processor:
- TOPIC (COTS)
- JMAPS (GOTS)

Desktop Publisher:
- FrameMaker (COTS)
- Interleaf (COTS)

Event Manager:
- CORBA (COTS)

User Interface (custom built):
- MOTIF version
- MS Windows version

4.3.1 Communicating Processes Architecture

In the communicating processes architecture depicted in Figure 1 (see [GS93]), components communicate directly to other components in a synchronous manner through remote procedure calls (RPCs); therefore there is no event manager component. The RPC mechanism is compliant with OSF's Distributed Computing Environment (DCE).

The Desktop Publishing component is optional in that not all command centers built from this architecture will include this component.

4.3.2 Event Systems Architecture

The event systems architecture depicted in Figure 2 uses an asynchronous event broadcast form of communication between components [GS93]. Most of the components communicate through an event manager component. The interfaces to the DBMS and user interface component are still via RPC. The event manager is a CORBA-compliant Object Request Broker [OMG]. The components use the CORBA dynamic invocation interface.

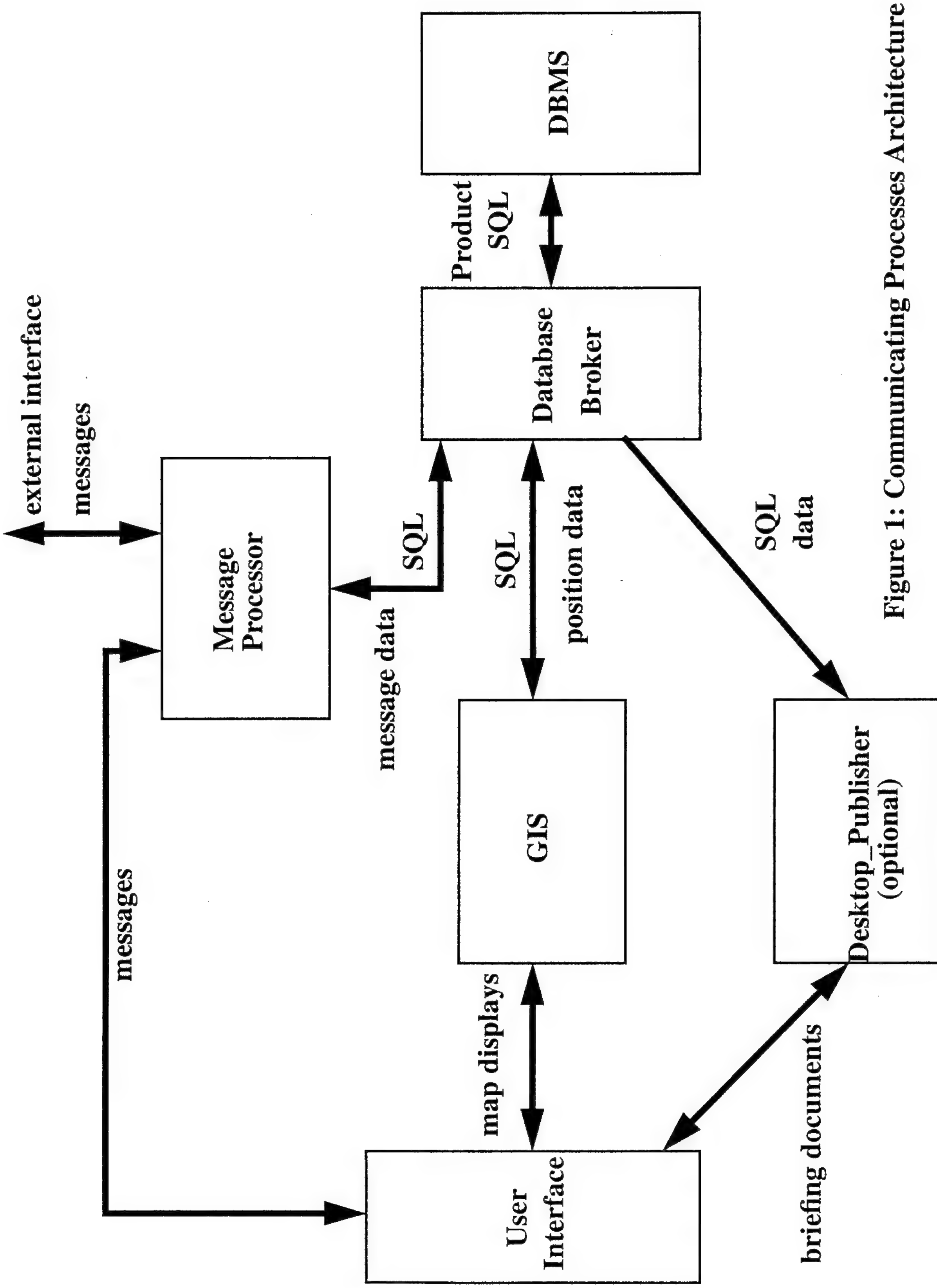


Figure 1: Communicating Processes Architecture

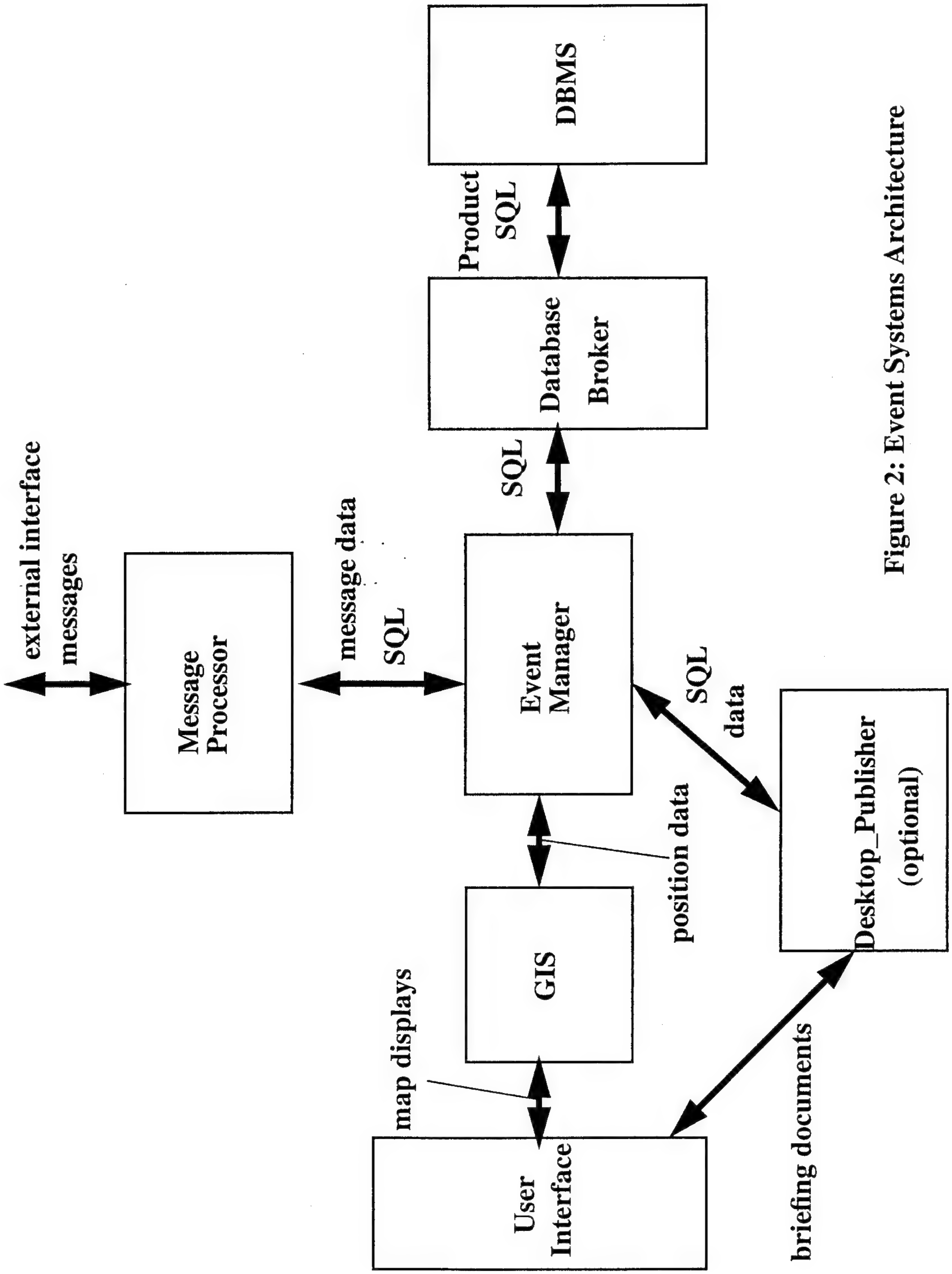


Figure 2: Event Systems Architecture

4.4 Tasks and Evaluation Questions

4.4.1 Domain Engineer

The domain engineer creates the architecture model.

4.4.1.1 Create Architecture Model for Command and Control System

Use the ADL to describe the communicating processes architecture shown in Figure 1 and all component instances listed in section 4.3. The ADL may be explicitly designed to support this style or the ADL can support this style but provides few or no built-in features to do so. Use all process support materials available including process definitions, users manuals, and training course materials. Use multiple views, if they are applicable in a particular ADL.

Complete the following parts of the ADL descriptive model framework evaluation form. Each column in the following tables indicates a level in the feature hierarchy.

Process-oriented (3.3)	**architecture creation support (3.3.1)**		**textual editor**
			graphical editor

	Process Support (3.3.7)		**process definition**
			users manual
			training course

System-oriented (3.1)	**applicability (3.1.1)**	**architecture style (3.1.1.1)**	**communicating processes**
			event systems

Language-oriented (3.2)	**language definition quality (3.2.1)**	**completeness (3.2.1.2)**	
		consistency (3.2.1.3)	
	Expressive power (3.2.2)	**primitives (3.2.2.1)**	
		extensibility (3.2.2.2)	
		abstractions (3.2.2.3 - see table after 3.2.8)	
	views (3.2.4)	**multiple views (3.2.4.1)**	
		syntactic views (3.2.4.2)	
		semantic views (3.2.4.3)	
		inter-view cross reference (3.2.4.4)	

		scalability (3.2.7.2)	**component size**

	variability (3.2.8)		**component instance**
			component inclusion

Evaluate the following usability features of the ADL (not in the current framework) on a scale from 1 to 5 (1 = Disagree .. 5 = Agree) [Christie94] [FS88]:

- The ADL was easy to learn.
- The ADL tools streamlines the process of creating an architecture description.
- The ADL is clear and consistent in its use of terminology.
- The ADL tools have a consistent "look and feel" in their user interface.
- The ADL tools display available options based on the current state.
- The ADL tools provide on-line help.
- The ADL tools provide useful error messages.

4.4.1.2 Modify Architecture Model

Modify the communicating processes architecture model that was created in step 4.4.1.1 so that it now describes the event systems architecture in Figure 2.

Complete the following parts of the ADL descriptive model framework evaluation form.

	modifiability (3.2.7)	**ease of change (3.2.7.1)**	

4.4.2 Application Engineer

The application engineer is the end user of the architecture description. These steps should be performed by someone who didn't do the steps in 4.4.1.

4.4.2.1 Component Qualification

Component qualification is the process of determining how well a new component "fits" into the architecture. By using only the information in the ADL, develop a list of criteria for qualifying a new GIS component.

Complete the following parts of the ADL descriptive model evaluation form:

	architecture refinement support (3.3.3)		**browser**
			search tool

	readability (3.2.5)	**embedded comments (3.2.5.1)**	
		presentation control (3.2.5.2)	
	Characteristics of intended users (3.2.6)	**Target users (3.2.6.1)**	
		Expertise (3.2.6.2)	

		system category (3.1.1.2)	**COTS and GOTS**
			distributed

Answer questions about the ADL from the application engineer perspective.

Evaluate the following usability features of the ADL (not in the current framework) on a scale from 1 to 5 (1 = Disagree .. 5 = Agree) [Christie94] [FS88]:

- The ADL screen layouts were easy to understand.
- The ADL was easy to learn.
- The icons for components were clearly distinguishable and easy to read.
- The icons for connections were clearly distinguishable and easy to read.
- The ADL makes effective use of color.
- The ADL is clear and consistent in its use of terminology.
- The ADL tools have a consistent "look and feel" in their user interface.
- The ADL tools display available options based on the current state.
- The ADL tools provide on-line help.

4.4.2.2 Application Building

Build an application system by refining the event system architecture using any tools available for the ADL. The optional desktop publisher component will be included in the application. Choose instances for each component to plug into the architecture. The result will be a detailed design derived from the architecture.

Complete the following parts of the ADL descriptive model framework evaluation form:

	architecture refinement support (3.3.3)		**incremental refinement**

	application building (3.3.5)		

References

[Binns93] Binns, Englehart, Jackson, Vestal. "Domain-Specific Software Architectures for Guidance, Navigation, and Control," Honeywell Technology Center, 1993.

[Booch86] Booch. "Object-Oriented Development," *IEEE Transactions on Software Engineering*, February 1986.

[BCC95] Brown, Carney, Clements. "A Case Study in Assessing the Maintainability of Large, Software-Intensive Systems," *Proceedings of Engineering of Computer Based Systems Conference*, Tucson AZ, March 1995.

[CARDS94] Comprehensive Approach to Reusable Defense Software (CARDS). "Technical Concepts Document," STARS-VC-B009/001/00, Unisys, January 1994.

[Christie94] Christie. "A Practical Guide to the Technology and Adoption of Software Process Automation," CMU/SEI-94-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, March 1994.

[CK95] Clements, Kogut. "Features of Architecture Description Languages," *Proceedings of the Seventh Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[DoD92] DoD Software Reuse Vision and Strategy, Document #1222-04-210/40, July 1992.

[FS88] Feiler, Smeaton. "The Project Management Experiment," CMU/SEI-88-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, July 1988.

[GS93] Garlan, Shaw. "An Introduction to Software Architecture," CMU/SEI-93-TR-33, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, December 1993. Also in *Advances in Software Engineering and Knowledge Engineering, Volume I*, Ambriola and Tortora (eds.), World Scientific Publishing, Singapore, 1993.

[KBAW94] Kazman, Bass, Abowd, Gregory, Webb. "SAAM: A Method for Analyzing the Properties of Software Architecture," *Proceedings for the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994.

[KR91] Kellner, Rombach. "Session Summary: Comparisons of Software Process Descriptions," *Proceedings of the 6th International Software Process Workshop: Support for the Software Process,* Hakodate, Hokkaido, Japan, July 1991.

[KC95] Kogut, Clements. "Features of Architecture Description Languages," draft SEI technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, December 1994.

[Krutchen94]Krutchen, Thompson. "An Object-Oriented, Distributed Architecture for Large Scale Ada Systems," *Proceedings of Tri-Ada 94*, November 1994.

[Luckham93]Luckham, Kenney, Augustin, Vera, Bryan, Mann. "Specification and Analysis of System Architecture Using Rapide," Stanford technical report, 1993.

[OMG91] "The Common Object Request Broker: Architecture and Specification," Object Management Group, Framingham MA, 1991.

[OSF93] "Introduction to OSF DCE," Open Software Foundation, Cambridge MA, 1993.

[PRISM93] Portable, Reusable, Integrated Software Modules (PRISM). "Generic Command Center Architecture Report for the Portable, Reusable, Integrated Software Modules Program," CDRL Sequence No. A001, Raytheon Company and Hughes Technical Services, May 1993.

[PWC] Parnas, Weiss, Clements. "The Modular Structure of Complex Systems," *Proceedings of the 8th International Conference on Software Engineering*, London, England, August 1985.

[Rombach91]Rombach. "A Framework for Assessing Process Representations," *Proceedings of the 6th International Software Process Workshop: Support for the Software Process,* Hakodate, Hokkaido, Japan, July 1991.

[Shaw94a] Shaw, Allen, Garlan, Klein, Ockerbloom, Scott, Schumacher. "Candidate Model Problems in Software Architecture," unpublished manuscript, version 1.2, November 1994.

[Shaw94b] Shaw. "Making Choices: A Comparison of Style for Software Architecture," unpublished manuscript, May 1994.

[Shaw94c] Shaw, DeLine, Klein, Ross, Young, Zelesnik. "Abstractions for Software Architectures and Tools to Support Them," unpublished manuscript, February 1994.

[Shirley93] Shirley. "Guide to Writing DCE Applications," O'Reilly & Associates, March 1993.

[SG88] Smith, Gerhart. "STATEMATE and Cruise Control: A Case Study," *Proceedings of COMPSAC 88*, 1988.

[STARS93] Software Technology for Adaptable, Reliable Systems (STARS). "Conceptual Framework for Reuse Processes (CFRP), Volume I: Definition, Version 3.0," STARS-VC-A018/001/00, Unisys, October 1993.

[Terry94] Terry, Hayes-Roth, Erman, Coleman, Devito, Papanagopoulos, Hayes-Roth. "Overview of Teknowledge's DSSA Program," *ACM SIGSOFT Software Engineering Notes*, October 1994.

[Tracz93] Tracz. "LILEANNA: A Parameterized Programming Language," *Proceedings of the 2nd International Workshop on Software Reuse*, March 1993.

[Weiderman86] Weiderman, Habermann, Borger, Klein. "A Methodology for Evaluating Environments," *Proceedings of the Second ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, Palo Alto CA, December, 1986.

Appendix A. ADL Evaluation: Capture

By providing a characterizing framework for individual ADLs, the descriptive model answers the following two questions about ADLs in general:

- What features do ADLs have?
- What features help describe the differences between ADLs?

The descriptive model is a framework for characterizing an individual ADL. A set of ADLs characterized in this framework can then be compared. The descriptive model contains a hierarchy of attributes that define important features of an ADL. For convenience, the descriptive model organizes attributes into the following broad categories:

- *System-oriented attributes* are attributes related to the application system derived from the software architecture that was encoded in the ADL. While all are attributes of the end system, they reflect on the ability of the ADL to produce such a system.
- *Language-oriented attributes* are attributes of the ADL itself, independent of the system(s) it is being used to develop. These attributes include the kind of information usually found in a language reference manual, if one exists.
- *Process-oriented attributes* are attributes of a process related to using an ADL to create, validate, analyze, and refine an architecture description, and build an application system.Included are attributes that measure or describe how or to what extent an ADL allows predictive evaluation of the application system with respect to that attribute. They are related to the semantics of the language that support analysis, evaluation, and verification of the architecture (Vestal 93). These attributes assert that the ADL contains enough information to make an architecture analyzable, whether or not tools actually exist that exploit the capability. In addition, the framework provides a place for existing tools to be described.

  For example, a language may allow enough timing information to be given to support schedulability analysis; a rate monotonic analysis schedule analyzer (if it exists for the language) would be an example of a tool that exploits such information.

  The analysis areas are primarily drawn from IEEE Std 1061, “Software Quality Metrics Methodology”. Many of these attributes are not addressed by any existing ADLs.

The descriptive model is articulated on the following pages in the form of a questionnaire, to be filled out with respect to a particular ADL. Some of the characteristics may overlap somewhat in meaning. The questionnaire provides space for each characteristic to be described, plus a free-form notes section where the respondent may elaborate his or her answers.

Notes: (**CC**) = results based on Cruise Control Scenario, (**C2**) = results based on Command & Control Scenario, otherwise results are composite.
(**NV**) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes

3.1.1 Applicability: How suitable is the ADL for representing a particular type of application system?

H = designed to support this style or type of system explicitly
M = could support this style or type of system but provides few or no built-in features to do so
L = does not support this style or type of system

3.1.1.1 Architecture style: How well does the ADL allow description of architectural styles, such as those enumerated in [Garlan93]? An architectural style is a family of architectures constrained by component/connector vocabulary, topology, and semantic constraints. Answer:

• Pipes and filters: linked stream transformers	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Main program and subroutines: traditional functional decomposition	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available.Structure Chart tool not available. (CC)
• Layered: structured layers with well-defined interfaces, and restrictions on cross-layer invocation	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Object-oriented: abstract data types with inheritance	H	Capture supports the Fusion method (Coleman) object-oriented representations. Representations are inherently graphic with textual annotation available. (CC)
• Communicating processes: synchronous or asynchronous message-passing, including client-server and peer-peer	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
• Event system: implicit invocation	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Transactional database system: central data repository, query driven	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Blackboard: central shared representation, opportunistic execution	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Interpreter: input driven state machine	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Rule based	M	Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)
• Heterogeneous styles	H	Capture can easily depict heterogeneous styles. (CC)
• Other styles (specify which)	M	state-based, real-time; Capture has the ability to simulate this style but does not directly support it. Representations are inherently graphic with textual annotation available. (CC)

3.1.1.2 Category: What broad classes of systems does the ADL support?

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
• **Hard real-time**: real-time systems in which the value of a result delivered any time past its deadline is zero, and may have catastrophic consequences	M	Real-time extensions to Data Flow Diagrams (e.g., explicit control flows, control stores), as depicted in Hatley-Pirbhai, Ward-Mellor methodologies are not available in off-the-shelf Capture product. State Transition Diagrams and Stimulus-Response Diagrams are useful for depicting behavioral views of time-critical systems. (CC)
• **Soft real-time**: real-time systems in which the value of a result delivered past its deadline decreases as a continuous function of the lateness	M	(see above) (CC)
• **Embedded**: systems that accept physical-world data from sensors and affect non-computer entities in the physical world	M	Stimulus-Response Diagrams appear particularly applicable to embedded systems. (see above) (CC)
• **Distributed**: systems implemented on separate processors such that communication cost is a substantial consideration	M	There are no special representations to support this system type. (CC)
• Dynamic architectures: systems in which architectural components are created and/or destroyed at run-time, or connections between components cannot be predicted before run-time.	H M L	
• Other categories (specify which)	H M L	
• **COTS/GOTS**	M	There are no special representations to support this system type. (C2)

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
3.1.1.3 Domains: What application domains is the ADL designed specifically to support, if any, and to what degree? List them.		
•	H M L	
•	H M L	
•	H M L	
•	H M L	
•	H M L	
•	H M L	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes

3.2.1 Language definition quality

3.2.1.1 Formality: How formally is the language defined? Answer:

informal, meaning no formal rules;
semi-formal, meaning formal rules exist for significant parts of the language;
formal, meaning formal rules exist that can be used to drive a formal reasoning system (e.g., a parser, or a theorem prover)
mix, meaning a combination of formal and informal). If mix, explain.

• Formality of syntax	*mix*	formal rules exist for the syntax of the architectural views, but these must be added to the off-the-shelf (OTS) product. **(NV)**
• Formality of semantics	*informal*	manual links establish relative semantics of representations

3.2.1.2 Completeness: How well is completeness defined for an architecture description? How does the ADL treat an incomplete architecture description?

H = major ADL design emphasis
M = supports to some degree
L = little or none

• Built-in completeness rules	L	None exist in the OTS Capture product (see below).
• User-specified completeness rules	M	Capture can be expanded to include rule-bases for given representation methods. **(NV)**
• Incompleteness toleration	L	Semantic checking does not exist in the OTS Capture product.
• Incompleteness marking (provided by token in language)	L	(see above)

Notes: **(CC)** = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.1.3 Consistency: Is self-consistency defined for an architecture description (internal)? Is it defined between two different architecture description, or between an architecture description and some other rendering of the system such as a requirements specification or coded implementation (external)? *H* = major ADL design emphasis *M* = supports to some degree *L* = little or none		
• Built-in internal consistency rules	L	Consistency is checked only with respect to the use of identifiers within a vertical asset family.
• User-defined internal consistency rules	L	These may be added. (NV)
• Built-in external consistency rules	L	Consistency is checked only with respect to the use of identifiers within a vertical asset family.
• User-defined external consistency rules	L	These may be added. (NV)

3.2.2 Expressive power of language		
3.2.2.1 Powerful primitives *H* = many useful built-in primitives *M* = some useful built-in primitives *L* = most useful primitives must be composed by user	H	Available primitives are those supported by the representation views supplied by the tool. Capture, as delivered, supports eight (8) views, none with any true tool-specific semantic checking.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.2.2 Extensibility *H* = major ADL design emphasis *M* = supports to some degree *L* = little or none		
• Ability to define new constructs in the language	H	Extensibility of non-textual constructs is addressed in cells below.
• Domain-specific extensions (describe)	H	Capture is non-domain-specific.
• Ability to define types of components, connectors, etc.	H	According to CTA, Capture is extensible, both in terms of additional views that can be provided, and the semantic checking that can be added to existing views. The "H" rating is given provisionally, subject to confirmation of this capability. **(NV)**

3.2.2.3 Abstractions: see separate table in Section 3.2.8

3.2.3 Scope of language: How much non-architecture information can the ADL represent? *H* = represents large amount *M* = represents significant amount *L* = represents little or none		
• Requirements	H M L	
• Detailed design/algorithms	H M L	

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Code	H M L	
• Test	H M L	
• Metrics	H M L	
• Other (specify which)	H M L	
3.2.4 Views: How well does the ADL support different views which highlight different aspects/perspectives of the architecture?		
3.2.4.1 Does the ADL support multiple views of the architecture?	Yes	Capture supports multiple architectural views at the same level. These include: Data Flow Diagram, (DFD)Entity-Relationship Diagram (ERD), State Transition Diagram (STD), Stimulus-Response Diagram (SRD), Fusion Object Interaction Graph, Fusion Visibility Graph, Fusion Inheritance Graph, and Fusion Object Model tools
3.2.4.2 Syntactic view list: Which syntactic views are supported?		
• Formal language	No	
• Free text	Yes	
• Graphical	Yes	
• Other (specify which, or indicate "none")	none	
3.2.4.3 Semantic view list: Which semantic views are supported?		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Data flow	Yes	Provided through Data Flow Diagram tool; data attribute description available through Entity-Relationship Diagram tool. (CC)
• Control flow	Yes	Provided implicitly through DFD Data Flows (to a certain extent, control flow [execution] is implied by data flow, although direction may not be equivalent); behavioral representation supported through State Transition Diagrams and Stimulus-Response Diagram tools. (see Section 3.1.1.2, **Hard Real-Time**) (CC)
• Process view (components are processes, connections are synchronizations)	Yes	Provided through DFD tool. (CC)
• State transition	Yes	Provided through STD tool. (CC)
• Other (specify which, or indicate "none")	Yes	Stimulus-Response Diagram (similar to *R-Nets* of the DCDS methodology and *process diagrams* of the RDD-100™ tool), Fusion Object Interaction Graph (shows how functionality is distributed across the objects of a system), Fusion Visibility Graph (show how an object-oriented system is structured to enable communication), Fusion Inheritance Graph (depict the inheritance structure of system objects in terms of *generalizations* and *specializations*, *superclasses*, *subclasses*, common *methods*, and *visibility*), and Fusion Object Model tools (based on Coleman's Fusion methodology — modified form of Entity-Relationship Diagram; its components are *class boxes* and *relationship diamonds*, connected by *subtype arcs* and *role arcs*)

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.4.4 Inter-view cross reference: Does the ADL provide for translating among views? *H* = extensive support; all views are interchangeable, meaning automatic inter-view translation is provided, and the same operations may be performed on any view *M* = some views are interchangeable (specify which) *L* = little or no support	L	Minimal translation support
3.2.5 Readability		
3.2.5.1 Embedded comments *H* = any construct may be commented *M* = comments may appear in certain places or with limitations *L* = commentary is not supported	H	Annotation capability — any <u>component</u> can be annotated (connectors cannot). Annotation occurs within a separate window that can be opened by selecting a component and the Annotation menu. All graphic entities can be labeled, but annotations exist via off-screen links.
3.2.5.2 Presentation control *H* = user has complete control over presentation, to enhance readability *M* = user has some control over presentation *L* = user has little or no control over presentation	M	User has little control over positioning of labels. Alignment and sizing of graphic entities lack precision. Overall, Capture has a adequate level of Presentation Control.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.6 Characteristics of intended users		
3.2.6.1 Target users		
• Domain engineer	Yes	
• Application engineer	Yes	
• Systems analyst	Yes	
• Software manager	Yes	Manager can determine necessary level of abstraction.

3.2.6.2 Expertise: What level of knowledge is needed to use the ADL effectively? Answer: *H* = considerable expertise required *M* = some expertise required *L* = little expertise required		
• Domain expertise (e.g., is the ADL domain-specific?)	L	Capture is non-domain-specific.
• Software design expertise (e.g., is any software design expertise built in to the ADL?)	M	
• Programming language expertise (e.g., is the ADL Ada-like?)	n/a	Capture is non-textual in nature — programming languages are note applicable to use of the tool.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.7 Modifiability of software architecture description		
3.2.7.1 Ease of change		
• Supports modularity *H* = change effects localized *M* = some rippling change effects *L* = change effects highly distributed	M	Local changes can be made quickly. Links to other components require more manual modifications.
3.2.7.2 Scalability: the degree to which the ADL can represent large and/or complex systems		
• Characterize the size of the largest project for which the ADL has been thoroughly used. Describe the number and size of components, and the complexity of their interaction.	Yes	Capture can be used to model large systems. (NV)
• abstraction levels: has the ability to represent hierarchical levels of detail	Yes	
• cross referencing: pointers to related information within the architecture description	No	
• subset capability: ability to partition the architecture description into smaller (more manageable) pieces that can be examined or analyzed in isolation	No	
• composition: ability to compose large pieces of the architecture into another architecture	No	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• multiple instantiation: the ability to make and insert copies of a component that are identical to the original, or vary in specified and systematic (e.g., parameterized) ways.	No	

3.2.8 Variability: How well does the ADL represent the variations in the application systems that can be derived from an architecture?

H = the variability is supported without having to change any but the replaced entities themselves
M = the variability is supported without having to change any but the replaced entities and those with which they directly interact
L = the user has to make many non-local changes to express variability

• Supports structural variability: adding or deleting components, or adding or deleting connections between existing components, or replacing components or connections with new ones of a different type.	H M L	
• Supports component variability: replacing a component with one that has a different functional interface or connection protocol.	H M L	
• Supports component instance variability: replacing a component with one that has the same functional interface and connection protocol.	H M L	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Supports component inclusion variability: representing variations in the exclusion or inclusion of components in a single architecture description, such as explicitly marking them as optional, or allowing the specification of alternative components.	H M L	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Abstractions What abstractions does the language support or provide? Name them. Characterize each by its type (2nd column) and by how appropriate it is to further understanding of the system at the architectural level (3rd column).	**Abstraction Type** C = component S = state or mode E = entity (node, data) CN = connector ST = state transition P = port R = role O = other	**Semantics of the abstraction** F = fixed by language; U = user-defined	**Appropriate** Y = yes N = no ? = unknown	**Notes**
Attribute	O	F		ERA entity characteristic
Class	O	F		used by Fusion Object Model; Object-Oriented classification entity
Collection	C	???		
Control Flow	CN	F		
Data Flow	CN	F		
Data Store	E	F		Data Flow Diagram static data entity
Entity	E	F		ERA data entity
Object	C	???		used by Fusion Object Interaction Graph

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Abstractions What abstractions does the language support or provide? Name them. Characterize each by its type (2nd column) and by how appropriate it is to further understanding of the system at the architectural level (3rd column).	**Abstraction Type** C = component S = state or mode E = entity (node, data) CN = connector ST = state transition P = port R = role O = other	**Semantics of the abstraction** F = fixed by language; U = user-defined	**Appropriate** Y = yes N = no ? = unknown	**Notes**
Process	C	F		used in both Data Flow and Stimulus-Response diagrams — similar meaning in both
Relationship	CN	F		used in E-R diagrams and Fusion Object Models
State	S	F		
Transition	ST	F		
Others (need more information):				
client/server, server collection, excl (?) server, excl server collection, perm vis, dynam ref, subclass, disjoint subclass	???			

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Abstractions What abstractions does the language support or provide? Name them. Characterize each by its type (2nd column) and by how appropriate it is to further understanding of the system at the architectural level (3rd column).	**Abstraction Type** C = component S = state or mode E = entity (node, data) CN = connector ST = state transition P = port R = role O = other	**Semantics of the abstraction** F = fixed by language; U = user-defined	**Appropriate** Y = yes N = no ? = unknown	**Notes**

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
3.3.1 Architecture creation support			
Textual editor: a tool (specific to the ADL) for directly manipulating textual descriptions of the architecture	N		
Graphical editor: a tool (specific to the ADL) for directly manipulating graphical descriptions of architectures	Y		
Import: a tool (specific to the ADL) to import information from other descriptions into the architecture	N		
3.3.2 Architecture validation support			
Syntax checker (e.g., parser)	N		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes

Support: Unless otherwise indicated, answer as follows:

Y = Not supported by any tool, but ADL contains enough information to perform the analysis
N = Not supported by information in the language.
E = supported by a tool external to the ADL.

? = unknown.

If supported by a tool specific to the ADL, name the tool;

Tool Type (if any)

S = simulation or prototype generation

A = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.

Notes

Semantics checker	N		Tool is extensible in this regard. **(NV)**
Completeness checker	N		Tool is extensible in this regard. **(NV)**
Internal consistency checker	N		Tool is extensible in this regard. **(NV)**
External consistency checker	N		

3.3.3 Architecture refinement support			
Browser: interactive support for undirected navigation of the description	Y		
Search tool: interactive support for directed navigation of the description	N		
Incremental refinement: interactive support for incrementally constraining design alternatives	N		

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Version control supported	N		
Architecture comparison: support for comparing two architecture representations to see if they represent the same architecture	N		Visual comparison only
3.3.4 Architecture analysis support			
3.3.4.1 Analyzing for time and resource economy			
Schedulability: degree to which the system meets its specific processing deadlines			
Throughput: the amount of work that can be performed by the system per unit time			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Other time economy attributes (specify)			
Memory utilization			
Other resource economy attributes (specify)			

3.3.4.2 Analyzing for functionality:			
Completeness: possessing necessary and sufficient functions to satisfy user needs			
Correctness: the degree to which software functions are satisfied			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 **Process-oriented attributes**	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Security: the degree to which software can detect or prevent information leakage, loss, or illegal use, or system resource destruction.			
Interoperability: the degree to which software can be connected easily with other systems and operated.			

3.3.4.3 Analyzing for maintainability			
Correctability: the degree of effort needed to correct errors in software			
Expandability: the degree of effort required to improve or modify the efficiency and functions of software			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

3.3 **Process-oriented attributes**	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Testability: the effort required to test software			
3.3.4.4 Analyzing for portability			
Hardware Independence: the degree to which software does not depend on specific hardware environments			
Software Independence: the degree to which software does not depend on specific software environments			
3.3.4.5 Analyzing for reliability			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Nondeficiency: the degree to which software does not contain undetected errors			
Error tolerance: the degree to which software will continue to work without a system failure that would cause damage to the users; also, the degree to which software includes degraded operation and recovery functions			
Availability: the degree to which software remains operable in the presence of system failures			

3.3.4.6 Analyzing for usability, the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system. Usability may be characterized (analyzed for) by observing a prototype or a simulation.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Understandability: the amount of user effort required to understand software			
Ease of learning: the degree to which user effort required to understand software is minimized			
Operability: the degree to which the operation of software matches the purpose, environment, and physiological characteristics of users, including ergonomic factors such as color, shape, sound, etc.			

3.3.5 Application building: building a compilable (or executable) software system from a specific system design			
3.3.5.1 System composition: the composition or integration of components for:			
Single processor target			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Distributed system with homogeneous processors and operating systems			
Components written in more than one programming language			
Distributed system with more than one variety of processors and/or operating systems			

3.3.5.2 Application generation support			
Component code generation			
Wrapper code generation			
Test case generation			
Documentation generation			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	Support: Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	Tool Type (if any) *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	Notes

3.3.6 Tool Maturity			
Commercial off-the-shelf (COTS)			
Age (years)			
Number of sites where tools are in use			
Customer support available by phone, email, or fax			
Maintenance support: are new versions released from time to time?			
Platforms: list the platforms on which the tools run			

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	Support: Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	Tool Type (if any) *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	Notes

3.3.7 Process support			
Process definition: Is there a documented sequence of steps for using the ADL? Answer *formal* (defined in a process modeling language or explained in detailed documentation), *informal* (specified by a high level process or example scenarios) *mix* (of formal and informal), or *none*	none		
Does a users manual exist?	yes		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Does a training course exist?	no		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Attachment 1 — ADL Usability Tables

These tables are not part of the current SEI ADL Feature Model, but are included here for the sake of completeness.

Table 1: ADL Usability — Domain Engineer Perspective

Statement	Rating (1 = Disagree 5 = Agree)	Notes
The ADL was easy to learn	5	The tool was easy to learn. A better manual is needed, as is some familiarity with the supported views and their associated methodologies. Example families and assets would also be helpful.
The ADL tools streamlines the process of creating an architecture description.	5	
The ADL is clear and consistent in its use of terminology.	4	Terminology varies according to the chosen view, but is consistent within context. The concept of "asset" may not be clear to some users.
The ADL tools have a consistent "look and feel" in their user interface.	5	
The ADL tools display available options based on the current state.	5	Capture is heavily state-based in its tool options.
The ADL tools provide on-line help.	3	Help is not always sufficient.
The ADL tools provide useful error messages.	2	The error messages in the product as evaluated could be improved.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Table 2: ADL Usability — End-User Perspective

Statement	Rating (1 = Disagree 5 = Agree)	Notes
The ADL screen layouts were easy to understand.		
The ADL was easy to learn.		
The icons for connections were clearly distinguishable and easy to read.		
The ADL makes effective use of color.		
The ADL is clear and consistent in its use of terminology.		
The icons for components were clearly distinguishable and easy to read.		
The ADL tools have a consistent "look and feel" in their user interface.		
The ADL tools display available options based on the current state.		
The ADL tools provide on-line help.		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Appendix B. ADL Evaluation: UniCon

By providing a characterizing framework for individual ADLs, the descriptive model answers the following two questions about ADLs in general:

- What features do ADLs have?
- What features help describe the differences between ADLs?

The descriptive model is a framework for characterizing an individual ADL. A set of ADLs characterized in this framework can then be compared. The descriptive model contains a hierarchy of attributes that define important features of an ADL. For convenience, the descriptive model organizes attributes into the following broad categories:

- *System-oriented attributes* are attributes related to the application system derived from the software architecture that was encoded in the ADL. While all are attributes of the end system, they reflect on the ability of the ADL to produce such a system.
- *Language-oriented attributes* are attributes of the ADL itself, independent of the system(s) it is being used to develop. These attributes include the kind of information usually found in a language reference manual, if one exists.
- *Process-oriented attributes* are attributes of a process related to using an ADL to create, validate, analyze, and refine an architecture description, and build an application system.Included are attributes that measure or describe how or to what extent an ADL allows predictive evaluation of the application system with respect to that attribute. They are related to the semantics of the language that support analysis, evaluation, and verification of the architecture (Vestal 93). These attributes assert that the ADL contains enough information to make an architecture analyzable, whether or not tools actually exist that exploit the capability. In addition, the framework provides a place for existing tools to be described.

  For example, a language may allow enough timing information to be given to support schedulability analysis; a rate monotonic analysis schedule analyzer (if it exists for the language) would be an example of a tool that exploits such information.

  The analysis areas are primarily drawn from IEEE Std 1061, "Software Quality Metrics Methodology". Many of these attributes are not addressed by any existing ADLs.

The descriptive model is articulated on the following pages in the form of a questionnaire, to be filled out with respect to a particular ADL. Some of the characteristics may overlap somewhat in meaning. The questionnaire provides space for each characteristic to be described, plus a free-form notes section where the respondent may elaborate his or her answers.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
3.1.1 Applicability: How suitable is the ADL for representing a particular type of application system? *H* = designed to support this style or type of system explicitly *M* = could support this style or type of system but provides few or no built-in features to do so *L* = does not support this style or type of system		
3.1.1.1 Architecture style: How well does the ADL allow description of architectural styles, such as those enumerated in [Garlan93]? An architectural style is a family of architectures constrained by component/connector vocabulary, topology, and semantic constraints. Answer:		
• Pipes and filters: linked stream transformers	H	Component and connector types are built-in.
• Main program and subroutines: traditional functional decomposition	H	Component and connector types are built-in.
• Layered: structured layers with well-defined interfaces, and restrictions on cross-layer invocation	M	Able to simulate layers with nested composite components. (not in either scenario - extrapolated answer)
• Object-oriented: abstract data types with inheritance	M	Able to specifiy abstract data types but does not support inherit-ance. Objects can be modules without global data access.
• Communicating processes: synchronous or asynchronous message-passing, including client-server and peer-peer	H	Component and connector types are built-in. **(C2)**
• Event system: implicit invocation	L	No built-in component types. Difficult to simulate implicit invo-cation with multiple RPC calls. **(C2)**
• Transactional database system: central data repository, query driven		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
• Blackboard: central shared representation, opportunistic execution		
• Interpreter: input driven state machine		
• Rule-based		
• Heterogeneous styles	H	Major design goal of UniCon
• Other styles: state-based/real-time	M	Can support rate monotonic analysis but does not support state transition diagrams.

3.1.1.2 Category: What broad classes of systems does the ADL support?		
• **Hard real-time**: real-time systems in which the value of a result delivered any time past its deadline is zero, and may have catastrophic consequences	H	Supports rate monotonic analysis
• **Soft real-time**: real-time systems in which the value of a result delivered past its deadline decreases as a continuous function of the lateness	H	
• **Embedded**: systems that accept physical-world data from sensors and affect non-computer entities in the physical world	M	Does not support complex behavioral analysis (e.g. state transitions)
• Distributed: systems implemented on separate processors such that communication cost is a substantial consideration	M	Does support RPCs but does not support network traffic simulation/analysis. (C2)

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.1 System-oriented attributes	Value	Notes
• Dynamic architectures: systems in which architectural components are created and/or destroyed at run-time, or connections between components cannot be predicted before run-time.	H M L	
• Other categories: COTS/GOTS architectures	H	Seems to scaleup to large preexisting components. **(C2)**
	H M L	

3.1.1.3 Domains: What application domains is the ADL designed specifically to support, if any, and to what degree? List them.		
•	H M L	
•	H M L	
•	H M L	
•	H M L	
•	H M L	
•	H M L	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes

3.2.1 Language definition quality

3.2.1.1 Formality: How formally is the language defined? Answer:

informal, meaning no formal rules;
semi-formal, meaning formal rules exist for significant parts of the language;
formal, meaning formal rules exist that can be used to drive a formal reasoning system (e.g., a parser, or a theorem prover)
mix, meaning a combination of formal and informal). If mix, explain.

• Formality of syntax	*formal*	explicit language definition
• Formality of semantics	*formal*	semantics are well defined

3.2.1.2 Completeness: How well is completeness defined for an architecture description? How does the ADL treat an incomplete architecture description?

H = major ADL design emphasis
M = supports to some degree
L = little or none

• Built-in completeness rules	Y	required information about components and connectors
• User-specified completeness rules	?	
• Incompleteness toleration	Y	Has optional attributes and general component type but seems to always require implementation information.
• Incompleteness marking (provided by token in language)	Y	Has general component type but incompleteness marking could be improved.

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.1.3 Consistency: Is self-consistency defined for an architecture description (internal)? Is it defined between two different architecture description, or between an architecture description and some other rendering of the system such as a requirements specification or coded implementation (external)? *H* = major ADL design emphasis *M* = supports to some degree *L* = little or none		
• Built-in internal consistency rules	Y	Many built-in component and connector types have player and role (inteface) rules. UniCon has strong support for catching interface inconsistencies at an architecture level.
• User-defined internal consistency rules	Y	optional attributes for players and roles
• Built-in external consistency rules	L	
• User-defined external consistency rules	L	

3.2.2 Expressive power of language		
3.2.2.1 Powerful primitives *H* = many useful built-in primitives *M* = some useful built-in primitives *L* = most useful primitives must be composed by user	H	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.2.2 Extensibility H = major ADL design emphasis M = supports to some degree L = little or none		
• Ability to define new constructs in the language	L	adding new basic component and connector abstraction is anticipated in the future
• Domain-specific extensions (describe)	M	can add restrictions on components/connectors reflecting the computing platform (e.g. Unix)
• Ability to define types of components, connectors, etc.	H	composite components and restrictions on connectors

3.2.2.3 Abstractions: see separate table in Section 3.2.8

3.2.3 Scope of language: How much non-architecture information can the ADL represent? H = represents large amount M = represents significant amount L = represents little or none		
• Requirements	H M L	
• Detailed design/algorithms	H M L	
• Code	H M L	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Test	H M L	
• Metrics	H M L	
• Other (specify which)	H M L	

3.2.4 Views: How well does the ADL support different views which highlight different aspects/perspectives of the architecture?		
3.2.4.1 Does the ADL support multiple views of the architecture?	Yes	
3.2.4.2 Syntactic view list: Which syntactic views are supported?		
• Formal language	Yes	
• Free text	No	
• Graphical	Yes	
• Other (specify which, or indicate "none")	none	
3.2.4.3 Semantic view list: Which semantic views are supported?		
• Data flow	Yes	not clear in all architecture styles (e.g., procedure calls)
• Control flow	Yes	implied by connector types
• Process view (components are processes, connections are synchronizations)	Yes	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• State transition	No	
• Other (specify which, or indicate "none")	none	
3.2.4.4 Inter-view cross reference: Does the ADL provide for translating among views? *H* = extensive support; all views are interchangeable, meaning automatic inter-view translation is provided, and the same operations may be performed on any view *M* = some views are interchangeable (specify which) *L* = little or no support	H	Automatic conversion between textual and graphical in both directions.

3.2.5 Readability		
3.2.5.1 Embedded comments *H* = any construct may be commented *M* = comments may appear in certain places or with limitations *L* = commentary is not supported	M	Comments can be embedded in textual notation only.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
3.2.5.2 Presentation control *H* = user has complete control over presentation, to enhance readability *M* = user has some control over presentation *L* = user has little or no control over presentation	H	Icons can be positioned by user. Component information can be displayed in pop-up windows. Graphical tools do not yet fully support creation of architecture models.
3.2.6 Characteristics of intended users		
3.2.6.1 Target users		
• Domain engineer	Yes	
• Application engineer	Yes	
• Systems analyst	Yes	
• Software manager	No	no management metrics supported

3.2.6.2 Expertise: What level of knowledge is needed to use the ADL effectively? Answer: *H* = considerable expertise required *M* = some expertise required *L* = little expertise required		
• Domain expertise (e.g., is the ADL domain-specific?)	L	UniCon is not domain-specific.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Software design expertise (e.g., is any software design expertise built in to the ADL?)	H	need to know about software design to understand component and connector types
• Programming language expertise (e.g., is the ADL Ada-like?)	M	using the textual notation is like programming

3.2.7 Modifiability of software architecture description		
3.2.7.1 Ease of change		
• Supports modularity *H* = change effects localized *M* = some rippling change effects *L* = change effects highly distributed	M	Need to worry about name clashes and inconsistencies. (originally part of C2 scenario but this was evaluated when creating the simulator part of the cruise control scenario)

3.2.7.2 Scalability: the degree to which the ADL can represent large and/or complex systems		
• Characterize the size of the largest project for which the ADL has been thoroughly used. Describe the number and size of components, and the complexity of their interaction.		
• abstraction levels: has the ability to represent hierarchical levels of detail	Yes	with composite components

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• cross referencing: pointers to related information within the architecture description	No	
• subset capability: ability to partition the architecture description into smaller (more manageable) pieces that can be examined or analyzed in isolation		
• composition: ability to compose large pieces of the architecture into another architecture	Yes	Very strong support for composing composite components into larger architectures.
• multiple instantiation: the ability to make and insert copies of a component that are identical to the original, or vary in specified and systematic (e.g., parameterized) ways.		

3.2.8 Variability: How well does the ADL represent the variations in the application systems that can be derived from an architecture?

H = the variability is supported without having to change any but the replaced entities themselves
M = the variability is supported without having to change any but the replaced entities and those with which they directly interact
L = the user has to make many non-local changes to express variability

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.2 Language-oriented attributes	Value	Notes
• Supports structural variability: adding or deleting components, or adding or deleting connections between existing components, or replacing components or connections with new ones of a different type.		
• Supports component variability: replacing a component with one that has a different functional interface or connection protocol.		
• Supports component instance variability: replacing a component with one that has the same functional interface and connection protocol.	H	Multiple implementation variants can be specified. **(C2)**
• Supports component inclusion variability: representing variations in the exclusion or inclusion of components in a single architecture description, such as explicitly marking them as optional, or allowing the specification of alternative components.	L	Did not find a way to do this. **(C2)**

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Abstractions What abstractions does the language support or provide? Name them. Characterize each by its type (2nd column) and by how appropriate it is to further understanding of the system at the architectural level (3rd column).	**Abstraction Type** C = component S = state or mode E = entity (node, data) CN = connector ST = state transition P = port R = role O = other	**Semantics of the abstraction** F = fixed by language; U = user-defined	**Appropriate** Y = yes N = no ? = unknown	**Notes**
module	C	F	Y	All UniCon abstractions have informative icons.
computation	C	F	Y	
SharedData	C	F	Y	
SeqFile	C	F	Y	
filter	C	F	Y	
process	C	F	Y	
SchedProcess	C	F	Y	
general	C	F	Y	
pipe	CN	F	Y	
FileIO	CN	F	Y	
ProcedureCall	CN	F	Y	

Notes: **(CC)** = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

Abstractions What abstractions does the language support or provide? Name them. Characterize each by its type (2nd column) and by how appropriate it is to further understanding of the system at the architectural level (3rd column).	**Abstraction Type** C = component S = state or mode E = entity (node, data) CN = connector ST = state transition P = port R = role O = other	**Semantics of the abstraction** F = fixed by language; U = user-defined	**Appropriate** Y = yes N = no ? = unknown	**Notes**
DataAccess	CN	F	Y	
RemoteProcCall	CN	F	Y	
RTScheduler	CN	F	Y	

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
3.3.1 Architecture creation support			
Textual editor: a tool (specific to the ADL) for directly manipulating textual descriptions of the architecture	Y		Use any standard ASCII editor. This task is similar to programming in an Ada-like language.
Graphical editor: a tool (specific to the ADL) for directly manipulating graphical descriptions of architectures	E		A powerful graphical editor has been developed but was not available for creation of the models for the scenario. Models were created in textual form then viewed/rearranged with the graphical tool.
Import: a tool (specific to the ADL) to import information from other descriptions into the architecture	E		A tool called *c2uni* converts C code into the UniCon textual notation.

3.3.2 Architecture validation support

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Syntax checker (e.g., parser)	E		error messages helpful but need improvement
Semantics checker	E		error messages helpful but need improvement
Completeness checker	E		error messages helpful but need improvement
Internal consistency checker	E		error messages helpful but need improvement
External consistency checker	N		

3.3.3 Architecture refinement support			
Browser: interactive support for undirected navigation of the description	E		Can navigate abstraction levels and get useful information in pop-up windows.
Search tool: interactive support for directed navigation of the description	Y		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Incremental refinement: interactive support for incrementally constraining design alternatives	Y		Can specify a processor and a component instance for each component in a refined system. But, there is no interactive support tool for refinement. (C2)
Version control supported			
Architecture comparison: support for comparing two architecture representations to see if they represent the same architecture			

3.3.4 Architecture analysis support

3.3.4.1 Analyzing for time and resource economy

Schedulability: degree to which the system meets its specific processing deadlines			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 **Process-oriented attributes**	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Throughput: the amount of work that can be performed by the system per unit time			
Other time economy attributes (specify)			
Memory utilization			
Other resource economy attributes (specify)			

3.3.4.2 **Analyzing for functionality:**			
Completeness: possessing necessary and sufficient functions to satisfy user needs			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Correctness: the degree to which software functions are satisfied			
Security: the degree to which software can detect or prevent information leakage, loss, or illegal use, or system resource destruction.			
Interoperability: the degree to which software can be connected easily with other systems and operated.			
3.3.4.3 Analyzing for maintainability			
Correctability: the degree of effort needed to correct errors in software			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Expandability: the degree of effort required to improve or modify the efficiency and functions of software			
Testability: the effort required to test software			

3.3.4.4 Analyzing for portability			
Hardware Independence: the degree to which software does not depend on specific hardware environments			
Software Independence: the degree to which software does not depend on specific software environments			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**

3.3.4.5 Analyzing for reliability			
Nondeficiency: the degree to which software does not contain undetected errors			
Error tolerance: the degree to which software will continue to work without a system failure that would cause damage to the users; also, the degree to which software includes degraded operation and recovery functions			
Availability: the degree to which software remains operable in the presence of system failures			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** S = simulation or prototype generation A = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
3.3.4.6 Analyzing for usability, the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system. Usability may be characterized (analyzed for) by observing a prototype or a simulation.			
Understandability: the amount of user effort required to understand software			
Ease of learning: the degree to which user effort required to understand software is minimized			
Operability: the degree to which the operation of software matches the purpose, environment, and physiological characteristics of users, including ergonomic factors such as color, shape, sound, etc.			

3.3.5 Application building: building a compilable (or executable) software system from a specific system design

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
3.3.5.1 System composition: the composition or integration of components for:			
Single processor target	E		need real components to test **(C2)**
Distributed system with homogeneous processors and operating systems	E		need real components to test **(C2)**
Components written in more than one programming language	?		
Distributed system with more than one variety of processors and/or operating systems	?		

3.3.5.2 Application generation support			
Component code generation	N		
Wrapper code generation	E		need real components to test **(C2)**

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Test case generation	N		
Documentation generation	N		

3.3.6 Tool Maturity			
Commercial off-the-shelf (COTS)			
Age (years)			
Number of sites where tools are in use			
Customer support available by phone, email, or fax			
Maintenance support: are new versions released from time to time?			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Platforms: list the platforms on which the tools run			

3.3.7 Process support			

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.

3.3 Process-oriented attributes	**Support:** Unless otherwise indicated, answer as follows: Y = Not supported by any tool, but ADL contains enough information to perform the analysis N = Not supported by information in the language. E = supported by a tool external to the ADL. ? = unknown. If supported by a tool specific to the ADL, name the tool;	**Tool Type (if any)** *S* = simulation or prototype generation *A* = analysis: based on reasoning, theory, and calculations without simulation (Vestal 93). E.g., verification.	**Notes**
Process definition: Is there a documented sequence of steps for using the ADL? Answer *formal* (defined in a process modeling language or explained in detailed documentation), *informal* (specified by a high level process or example scenarios) *mix* (of formal and informal), or *none*	informal		Examples are given in Shaw et. al. 94.
Does a users manual exist?	yes		Unix-style man pages are available. Shaw, et al, 94 explains the concepts well.
Does a training course exist?	no		

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

Attachment 1 — ADL Usability Tables

These tables are not part of the current SEI ADL Feature Model, but are included here for the sake of completeness.

Table 1: ADL Usability — Domain Engineer Perspective

Statement	Rating (1 = Disagree 5 = Agree)	Notes
The ADL was easy to learn	3	You need to understand the concepts before you can use UniCon. The concepts are not difficult if you have broad knowledge of software design.
The ADL tools streamlines the process of creating an architecture description.	?	The graphical tool was not available for creation of the architecture models. The parser for the textual view works well.
The ADL is clear and consistent in its use of terminology.	4	There is a large amount of terminology involved.
The ADL tools have a consistent "look and feel" in their user interface.	5	
The ADL tools display available options based on the current state.	3	Needs improvement -- still maturing.
The ADL tools provide on-line help.	1	None available yet.
The ADL tools provide useful error messages.	4	Most of the time.

Notes: (CC) = results based on Cruise Control Scenario, (C2) = results based on Command & Control Scenario, otherwise results are composite. (NV) = not verified, based on information supplied by vendor/creator.

Table 2: ADL Usability — Application Engineer Perspective

Statement	Rating (1 = Disagree 5 = Agree)	Notes
The ADL screen layouts were easy to understand.		
The ADL was easy to learn.		
The icons for connections were clearly distinguishable and easy to read.		
The ADL makes effective use of color.		
The ADL is clear and consistent in its use of terminology.		
The icons for components were clearly distinguishable and easy to read.		
The ADL tools have a consistent "look and feel" in their user interface.		
The ADL tools display available options based on the current state.		
The ADL tools provide on-line help.		

Notes: (CC) = results based on Cruise Control Scenario, **(C2)** = results based on Command & Control Scenario, otherwise results are composite.
(NV) = not verified, based on information supplied by vendor/creator.